PATENT ABSTRACTS OF JAPAN

(11)Publication number : 09-212359
(43)Date of publication of application : 15.08.1997

(51)Int.CI. G06F 9/38

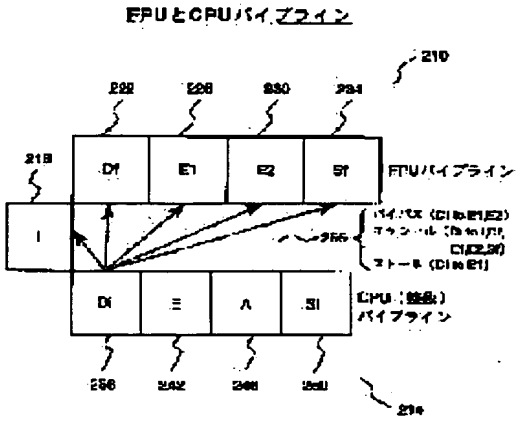
(21)Application number : 08-249650
(22)Date of filing : 20.09.1996

(71)Applicant : HITACHI LTD
(72)Inventor : PURASENJITSUTO BISUWASU
GAUTAMU DEWAN
KEBIN IADONATO
NAKAGAWA NORIO
UCHIYAMA KUNIO

(30)Priority
Priority number : 96 594763 Priority date : 31.01.1996 Priority country : US

(54) MICROPROCESSOR AND DATA PROCESSING METHOD

(57)Abstract:
PROBLEM TO BE SOLVED: To reduce the physical circuit scale and to attain synchronization between an FPU pipeline and a CPU pipeline.
SOLUTION: The microprocessor executes an instruction by advancing the FPU pipeline 210 and the CPU pipeline 214 in parallel. When the number of stages in both the pipelines 210, 214 are basically set to the same number, resources for pipeline control such as the flow control of both the pipelines 210, 214 can be shared. When freezing or stalling is necessary for the pipelines 210, 214, one of the pipelines 210, 214 controls freezing or stalling so as to bring stalling and freezing effects to both the pipelines 210, 214, so that synchronization between the FPU pipeline 210 and the CPU pipeline 214 can be attained.



LEGAL STATUS

[Date of request for examination] 26.02.2003

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平9－212359

(43)公開日　平成９年(1997)８月15日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/38	３１０		Ｇ０６Ｆ　9/38	３１０Ｊ	
				３１０Ｅ	

審査請求　未請求　請求項の数８　ＯＬ　（全 51 頁）

(21)出願番号　特願平8－249650

(22)出願日　平成８年(1996)９月20日

(31)優先権主張番号　５９４７６３

(32)優先日　1996年１月31日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地

(72)発明者　プラセンジット　ビスワス
アメリカ合衆国　カリフォルニア州　95070、サラトガ、バンパス　コート　20167

(72)発明者　ガウタム　デワン
アメリカ合衆国　カリフォルニア州　95014、クーパーティノ、ホームステッド　ロード　20800　エーピーティー　#15　ケー

(74)代理人　弁理士　玉村　静世

最終頁に続く

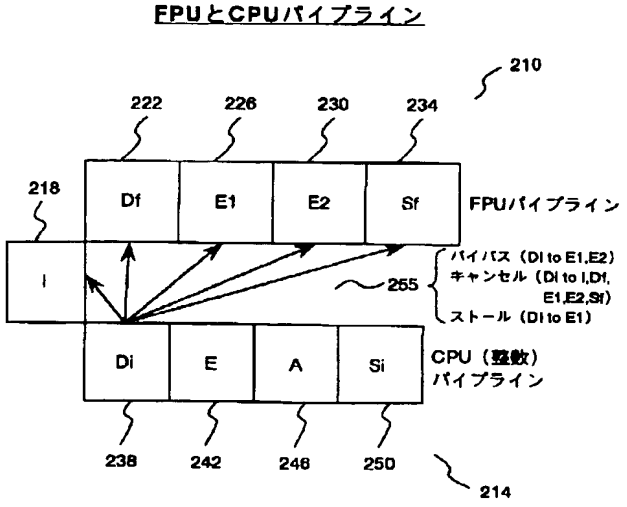
(54)【発明の名称】　マイクロプロセッサ及びデータ処理方法

(57)【要約】

【課題】　物理的な回路規模を縮小してＦＰＵパイプラインとＣＰＵパイプラインとを同期化する。

【解決手段】　マイクロプロセッサは、ＦＰＵパイプライン（２１０）とＣＰＵパイプライン（２１４）を並列的に進めて命令を実行する。双方のパイプライン段数を基本的に同一とすることにより、双方のパイプラインのフロー制御などのパイプライン制御のためのリソースを共有化する。パイプラインに対してフリーズやストールを必要とする場合、双方のパイプラインにストールとフリーズの効果をもたらすようにパイプラインの一方がストール又はフリーズを制御することなどによってＦＰＵパイプラインとＣＰＵパイプラインの同期化を達成する。

【図２】



【特許請求の範囲】

【請求項１】　浮動小数点処理を実行する浮動小数点パイプラインと整数処理及びメモリアドレシング処理を実行する整数パイプラインとを用いて命令を実行する浮動小数点ユニット及び中央処理装置を含んで成るマイクロプロセッサであって、

前記浮動小数点パイプラインのパイプライン段数と前記整数パイプラインのパイプライン段数とが等しくされて成るものであることを特徴とするマイクロプロセッサ。

【請求項２】　前記浮動小数点パイプライン及び整数パイプラインは共通の命令フェッチステージを有し、前記浮動小数点パイプラインは更に、デコードステージ、第１の実行ステージ、第２に実行ステージ及びライトバックステージを有し、前記整数パイプラインは更に、デコードステージ、実行ステージ、メモリアクセスステージ及びライトバックステージを有して成るものであることを特徴とする請求項１記載のマイクロプロセッサ。

【請求項３】　前記整数パイプラインのデコードステージは、浮動小数点パイプライン及び整数パイプラインのためのストール信号を生成するものであることを特徴とする請求項１記載のマイクロプロセッサ。

【請求項４】　前記浮動小数点ユニットと中央処理装置は、浮動小数点パイプラインと整数パイプラインとのフロー処理を制御する論理を共有するものであることを特徴とする請求項１記載のマイクロプロセッサ。

【請求項５】　浮動小数点ユニットと中央処理装置とを含むマイクロプロセッサにおいて、浮動小数点処理を実行する浮動小数点パイプラインと整数及びメモリアドレシング処理を実行する整数パイプラインとを用いて命令を実行するデータ処理方法であって、

前記浮動小数点パイプラインのパイプライン段数は前記整数パイプラインのパイプライン段数に等しくされ、

前記浮動小数点パイプラインと整数パイプラインは共通の命令フェッチステージを有し、

中央処理装置のレジスタから浮動小数点ユニットのレジスタにデータを転送する第１の命令を前記命令フェッチステージでフェッチする処理と、

整数パイプラインが前記第１の命令をデコードして中央処理装置のレジスタの値をバスに出力する処理と、

浮動小数点パイプラインが前記第１の命令をデコードして、前記出力する処理と同期的に前記バスの値を浮動小数ユニットのレジスタに書き込む処理と、を含むことを特徴とするデータ処理方法。

【請求項６】　浮動小数点ユニットと中央処理装置とを含むマイクロプロセッサにおいて、浮動小数点処理を実行する浮動小数点パイプラインと整数及びメモリアドレシング処理を実行する整数パイプラインとを用いて命令を実行するデータ処理方法であって、

前記浮動小数点パイプラインのパイプライン段数は前記整数パイプラインのパイプライン段数に等しくされ、

前記浮動小数点パイプラインと整数パイプラインは共通の命令フェッチステージを有し、

浮動小数点ユニットのレジスタから中央処理装置のレジスタにデータを転送する第２の命令を前記命令フェッチステージでフェッチする処理と、

浮動小数点パイプラインが前記第２の命令をデコードして浮動小数点ユニットのレジスタの値をバスに出力する処理と、

整数パイプラインが前記第２の命令をデコードして、前記出力する処理と同期的に前記バスの値を中央処理装置のレジスタに書き込む処理と、を含むことを特徴とするデータ処理方法。

【請求項７】　浮動小数点ユニットと中央処理装置とを含むマイクロプロセッサにおいて、浮動小数点処理を実行する浮動小数点パイプラインと整数及びメモリアドレシング処理を実行する整数パイプラインとを用いて命令を実行するデータ処理方法であって、

前記浮動小数点パイプラインのパイプライン段数は前記整数パイプラインのパイプライン段数に等しくされ、

前記浮動小数点パイプラインと整数パイプラインは共通の命令フェッチステージを有し、前記浮動小数点パイプラインは更に、デコードステージ、第１の実行ステージ、第２に実行ステージ及びライトバックステージを有し、前記整数パイプラインは更に、デコードステージ、実行ステージ、メモリアクセスステージ及びライトバックステージを有し、

比較結果の一致／不一致に応じた論理値のＴビットを生成する浮動小数点比較命令に対して浮動小数点パイプラインが前記第１の実行ステージでＴビットを生成する処理と、

Ｔビットの値を分岐条件とする条件分岐命令に対して整数パイプラインがその実行ステージで前記Ｔビットを参照する処理と、を含むことを特徴とするデータ処理方法。

【請求項８】　浮動小数点ユニットと中央処理装置とを含むマイクロプロセッサにおいて、浮動小数点処理を実行する浮動小数点パイプラインと整数及びメモリアドレシング処理を実行する整数パイプラインとを用いて命令を実行するデータ処理方法であって、

前記浮動小数点パイプラインのパイプライン段数は前記整数パイプラインのパイプライン段数に等しくされ、

前記浮動小数点パイプラインと整数パイプラインは共通の命令フェッチステージを有し、前記浮動小数点パイプラインは更に、デコードステージ、第１の実行ステージ、第２に実行ステージ及びライトバックステージを有し、前記整数パイプラインは更に、デコードステージ、実行ステージ、メモリアクセスステージ及びライトバックステージを有し、

前記第１の実行ステージを複数回繰り返す浮動小数点除算命令を命令フェッチステージでフェッチする処理と、

フェッチされた前記浮動小数点除算命令における第１の実行ステージの期間にビジー信号をイネーブルにする処理と、
前記ビジー信号がイネーブルにされている期間において、浮動小数点パイプラインを利用する後続の命令の第１の実行ステージ以降のパイプラインステージと整数パイプラインのステージとを共に同期させてストールする処理と、を含むことを特徴とするデータ処理方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、浮動小数点ユニット及び中央処理装置を含んだマイクロプロセッサ、特にそれにおけるパイプライン制御技術に関し、例えば、ＲＩＳＣ形式のマイクロプロセッサに適用して有効な技術に関するものである。

【０００２】

【従来の技術】あるＲＩＳＣ（縮小命令セットコンピュータ）マイクロプロセッサは、ＦＰＵ（浮動小数点ユニット）を備える。浮動小数点ユニットは、浮動小数点計算を実行する回路である。ＲＩＳＣは、相対的に単純で、固定サイズの命令を用いて、マイクロプロセッサの複雑さを減少させたコンピュータアーキテクチャである。ＲＩＳＣアーキテクチャにおける殆どの命令は、汎用レジスタを利用してオペランドの操作を行ない、結果をレジスタにストアする。これらのレジスタは、メモリからロードされ、典型的には、レジスタの内容はプログラムの実行中に再利用される。殆どのＲＩＳＣアーキテクチャは１６又はそれ以上の汎用レジスタを持つ。

【０００３】

【発明が解決しようとする課題】典型的なＲＩＳＣマイクロプロセッサは、パイプラインで命令を実行する能力を持つ。多数の機能ユニット（ＣＰＵの整数処理などを実行するＣＰＵパイプラインや浮動小数点処理を実行するＦＰＵパイプライン）の動作を調整することにおいて多数の問題がある。そのような装置における二つのユニット（ＣＰＵパイプラインを制御するユニットとＦＰＵパイプラインを制御するユニット）がパイプライン制御のためのリソースを共有するなら、二つのパイプラインの動作を同期化させることは、その問題の解決に大きな役割を演ずるであろうことが本発明者によって明らかにされた。すなわち、ＦＰＵとＣＰＵを含むマクロプロセッサについて考える。従来、ＦＰＵパイプライン及びＣＰＵパイプラインの夫々の段数（パイプラインステージの数）は相違されている。浮動小数点演算は整数演算に比べて演算時間がかかるので、実行ステージに３段のパイプラインステージを割り当てたものがある。ＦＰＵパイプラインとＣＰＵパイプラインの段数が相違されると、ストールやバイパスなどのフロー制御を行うためのデコードロジック等はＣＰＵ及びＦＰＵ双方のパイプライン毎に別々に構成することが必要になる。例えば命令毎のパイプライン間でのバイパスを一例として考えると、前後の命令でレジスタコンフリクトのようなデータコンフリクトを生ずるとき、前の命令が演算を完了したとき、その実行ステージの結果をバイパス経路にて後の命令の実行ステージに直接与えることによって当該後の命令は先の命令の実行完了を待つことなくオペランドを得ることができるが、どのステージからどのステージにバイパスさせるべきかは、パイプラインステージの段数に依存して決定されなければならない。このとき、ＦＰＵパイプラインとＣＰＵパイプラインの長さ（パイプライン段数）が相違すれば、バイパス制御などのフロー制御のためのリソースもＣＰＵパイプラインとＦＰＵパイプラインとの間で殆ど個別化することが必要になり、また、ＦＰＵパイプラインとＣＰＵパイプラインとの間でのデータ交換に際しても複雑なハードウェアが必要になってしまう。本発明者はそれら点について見出した。

【０００４】もう一つの問題は、正確に例外の意味を維持することである。パイプライン化されたアーキテクチャ又はマルチ・ファンクションユニット・アーキテクチャにおいて例外又は割り込みを正確に（プログラムの実行順序を乱さずに）処理することは、例外又は割り込みが発生したとき、アーキテクチャ上連続的に実行されるプログラムと正確に同じになるように、マシンの状態をセーブ可能であるべきことを意味する。たとえ中継ユニットに指示を発行する命令が厳密にプログラムの順序を維持するとしても、命令の実行順序は、異なった機能ユニットでは命令の実行時間が異なるため、乱れる。例えば、ＣＰＵパイプライン及びＦＰＵパイプラインを有するプロセッサにおいて、浮動小数点除算命令のような通常よりも実行サイクル数の多い命令をＦＰＵパイプラインで処理するとき、その処理で０除算のような例外が発生したとき、当該浮動小数点除算命令の後に続く命令の実行状態如何によっては、例外発生に起因して当該浮動小数点除算命令を再実行すると、命令実行順序がプログラムで規定されている命令実行順序を乱して、制御が複雑になる場合がある。

【０００５】パイプライン化されたプロセッサにおいて上記の意味で正確に割り込みを実施するための効果的な手段は、先行技術としてのＩＥＥＥコンピュータ処理，第５６２～５７３頁の「パイプライン化されたプロセッサにおける厳密な割り込み」（“Implementing Precise Interrupts in Pipelined Processor” IEEE Transaction on Computers, pp. 562-573 , May 1988））で議論されている。現在のパイプライン化された多くの機能ユニットを有するプロセッサの殆どは、そこで言及されている技術に基づいている。

【０００６】それら技術の内のあるものは、レジスタファイルの追加を必要とし、制御のための複雑なロジックを必要とする。リソースを共有して同期化するには、共有リソースに結合された更に複雑な内部データバスと同

様に、タグ・マッチングのハードウェアを必要とする。他の技術は、レジスタリソースのコンフリクト（競合）を識別し且つ解消するためのレジスタスコアボードを使用する。これらの技術は、要するに、チップエリアの追加を要求し、エンベッデット・アプリケーション（機器組み込み制御）を予定する低価格のプロセッサには適していない。

【０００７】本発明の目的は、浮動小数点パイプラインと整数パイプラインとを同期化させることができるマイクロプロセッサを提供することにある。

【０００８】更に本発明は、上記同期化に際してパイプライン制御のためのリソースを低減出来るマイクロプロセッサを提供することにある。

【０００９】本発明の別の目的は、ＣＰＵとＦＰＵとの間のデータ交換を簡単にしかも高速に行なうことができるデータ処理方法を提供することにある。

【００１０】本発明の更に別の目的は、浮動小数点比較命令の後に続く条件分岐に対するスループットを向上させることができるデータ処理方法を提供することにある。

【００１１】本発明のその他の目的は、浮動小数点除算命令においても浮動小数点パイプラインと整数パイプラインとを同期化できるデータ処理方法を提供することにある。

【００１２】本発明の前記並びにその他の目的と新規な特徴は本明細書の記述及び添付図面から明らかになるであろう。

【００１３】

【課題を解決するための手段】本願において開示される発明のうち代表的なものの概要を簡単に説明すれば下記の通りである。

【００１４】すなわち、　マイクロプロセッサ（１１０）は、浮動小数点処理を実行する浮動小数点パイプライン（ＦＰＵパイプライン）、そして、整数計算及びメモリアドレス操作を実行する整数パイプライン（ＣＰＵパイプライン）を利用して命令を実行する。すなわち、命令実行に際して、ＦＰＵパイプラインとＣＰＵパイプラインを並列的に進めていく。浮動小数点パイプライン（２１０）とＣＰＵパイプライン（２１４）とのパイプライン段数（パイプラインの長さ）を基本的に同一とすることによって、ＣＰＵパイプラインとＦＰＵパイプラインのフロー制御などのパイプライン制御のためのリソースの共有化を図る。これによって、浮動小数点パイプラインと整数パイプラインとを同期化し易くする。パイプラインに対してフリーズやストールを必要とする場合、双方のパイプラインにストールとフリーズの効果をもたらすようにパイプラインの一方がストール又はフリーズを制御することによってＦＰＵパイプラインとＣＰＵパイプラインの同期化を達成する。

【００１５】例外は、長い浮動小数点処理でも正確に保たれる。正確な例外は、プログラムが規定する命令実行順序における先の浮動小数点命令である浮動小数点除算命令がＦＰＵパイプラインの最初の実行ステージに入るとき、ＦＰＵパイプラインステージの最初の実行ステージでビジー信号をアサートする（イネーブルにする）ことによって達成される。前記先の浮動小数点命令がＦＰＵパイプラインの最初の実行ステージを繰返し利用する複数ステージ分の時間を費やした実行を完了する前に、後の浮動小数点命令がＦＰＵパイプラインによってデコードされる場合、それ以降のＣＰＵ及びＦＰＵの双方のパイプラインがストールされる。

【００１６】本発明による手段を更に説明する。

【００１７】浮動小数点処理を実行する浮動小数点パイプライン（２１０）と整数処理及びメモリアドレシング処理を実行する整数パイプライン（２１４）とを用いて命令を実行する浮動小数点及び中央処理装置を含むマイクロプロセッサは、前記浮動小数点パイプラインのパイプライン段数と前記整数パイプラインのパイプライン段数とが等しくされて成る。双方のパイプライン（２１０，２１４）の段数が等しければ、ストール及びフリーズを制御するフロー制御等のためのリソースを双方のパイプラインで共有させることができる。これは、チップ面積の縮小を可能にする。フロー制御のためのリソースの共有は双方のパイプラインの同期化を容易にする。

【００１８】前記整数パイプラインのデコードステージは、浮動小数点パイプライン及び整数パイプラインのためのストール信号を生成することができる。

【００１９】浮動小数点ユニット及び中央処理装置を含むマイクロプロセッサにおいて、浮動小数点処理を実行する浮動小数点パイプラインと整数処理及びメモリアドレシング処理を実行する整数パイプラインとを用いて命令を実行するデータ処理方法の観点では、前記浮動小数点パイプラインのパイプライン段数は前記整数パイプラインのパイプライン段数に等しくされ、前記浮動小数点パイプラインと整数パイプラインは共通の命令フェッチステージを有する。このとき、ある命令を実行するときにおけるＦＰＵパイプラインとＣＰＵパイプラインとの同期化の例としては、中央処理装置のレジスタから浮動小数点ユニットのレジスタ（ＦＰＵＬ）にデータを転送する第１の命令によるデータ処理方法がある。この第１の命令を実行するデータ処理方法は、この命令を前記命令フェッチステージでフェッチする処理と、整数パイプラインが前記第１の命令をデコードして中央処理装置のレジスタの値をバスに出力する処理と、浮動小数点パイプラインが前記第１の命令をデコードして前記出力する処理と同期的に前記バスの値を浮動小数ユニットのレジスタに書き込む処理とを含む。

【００２０】また、上記同期化の別の例として、浮動小数点ユニットのレジスタから中央処理装置のレジスタにデータを転送する第２の命令によるデータ処理方法を挙

げることができる。この第２の命令を実行するデータ処理方法は、当該第２の命令を前記命令フェッチステージでフェッチする処理と、浮動小数点パイプラインが前記第２の命令をデコードして浮動小数点ユニットのレジスタ（ＦＰＵＬ）の値をバスに出力する処理と、整数パイプラインが前記第２の命令をデコードして前記出力する処理と同期的に前記バスの値を中央処理装置のレジスタに書き込む処理とを含む。

【００２１】前記浮動小数点パイプラインのパイプライン段数を前記整数パイプラインのパイプライン段数に等しくし、前記浮動小数点パイプラインと整数パイプラインが共通の命令フェッチステージを有する場合の前記データ処理方法において、前記浮動小数点パイプラインは更に、デコードステージ、第１の実行ステージ、第２に実行ステージ及びライトバックステージを有し、前記整数パイプラインは更に、デコードステージ、実行ステージ、メモリアクセスステージ及びライトバックステージを有するとき、浮動小数点比較命令に後続する条件分岐命令による条件分岐のスループットを向上させることを企図したデータ処理方法は、比較結果の一致／不一致に応じた論理値のＴビットを生成する浮動小数点比較命令に対して浮動小数点パイプラインが第１の実行ステージでＴビットを生成する処理と、Ｔビットの値を分岐条件とする条件分岐命令に対して整数パイプラインがその実行ステージで前記Ｔビットを参照する処理とを含む。これによって、浮動小数点比較命令の直後に条件分岐命令が配置されている場合にストールは一切不要である。

【００２２】前記浮動小数点パイプラインのパイプライン段数を前記整数パイプラインのパイプライン段数に等しくし、前記浮動小数点パイプラインと整数パイプラインが共通の命令フェッチステージを有する場合のデータ処理方法において、前述と同じく、前記浮動小数点パイプラインは更にデコードステージ、第１の実行ステージ、第２に実行ステージ及びライトバックステージを有し、前記整数パイプラインは更にデコードステージ、実行ステージ、メモリアクセスステージ及びライトバックステージを有するとき、浮動小数点除算命令によるＦＰＵパイプラインとＣＰＵパイプラインとの同期化を保つようにするデータ処理方法は、第１の実行ステージを複数回繰り返す浮動小数点除算命令を命令フェッチステージでフェッチする処理と、フェッチされた前記浮動小数点除算命令における第１の実行ステージの期間にビジー信号をイネーブルにする処理と、前記ビジー信号がイネーブルにされている期間において、浮動小数点パイプラインを利用する後続の命令の第１の実行ステージ以降のパイプラインステージと整数パイプラインのステージとを共に同期させてストールする処理とを含む。

【００２３】

【発明の実施の形態】本発明の好適な例において、マイクロプロセッサは、以下に説明されるように、浮動小数点処理を実行するために浮動小数点パイプライン（ＦＰＵパイプライン）２１０（図２参照）を利用し、整数及びメモリアドレシング処理などを実行するために整数パイプライン（ＣＰＵパイプライン）２１４（図２参照）を利用する。ＦＰＵパイプライン２１０とＣＰＵパイプライン２１４とは並列的に進行され、その過程で同期化が行われる。ＦＰＵパイプライン２１０とＣＰＵパイプライン２１４のパイプライン段数（パイプラインのステージ数）を相互に等しくしてある。これによって、ＣＰＵパイプライン２１４とＦＰＵパイプライン２１０とのフロー制御等のためのパイプライン制御用のリソースがＦＰＵパイプラインとＣＰＵパイプラインで殆ど共有出来るようになる。このパイプラインリソースの共有は、ＣＰＵパイプライン２１４とＦＰＵパイプライン２１０との同期化を容易にする。

【００２４】ＦＰＵパイプライン２１０は幾つかの点でＣＰＵパイプライン２１４に同期化される。ＦＰＵ及びＣＰＵパイプライン２１０，２１４は互いに同期化され、例えば、浮動小数点のロード／ストア／リストア命令のような命令が、ＦＰＵ１１４とＣＰＵ１１８がリソース（例えばコミュニケーションレジスタ）を共有するときである。ＦＰＵとＣＰＵパイプライン２１０，２１４は、それらがコミュニケーションレジスタＦＰＵＬを使ってデータ交換を行なうとき同期化される。また、命令の手順が、例えばＣＰＵ分岐命令が後に続く浮動小数点比較命令においても、パイプラインの同期化が行われる。これら二つのパイプライン２１０，２１４の同期化は、更に以下で説明するように、正確な例外を保つのにも有効である。

【００２５】図１は本発明の好適な一例に係るプロセッサのブロックダイヤグラムを示す。プロセッサ１１０は浮動小数点ユニット（ＦＰＵ）１１４を持つ。更に、プロセッサ１１０は、整数を操作することができる中央処理装置（ＣＰＵ）１１８を備える。プロセッサ１１０は、１６ビット固定長浮動小数点命令セットを備えた３２ビットＲＩＳＣアーキテクチャとされる。ＣＰＵ１１８のための１６ビット命令の詳細は、論文、すなわち“ＳＨ３、高コード密度及びローパワー”ＩＥＥＥＭｉｃｒｏ，第１１～１９、１２月、１９９５（"SH3:High Code Density, Low Power", IEEE Micro, pp.11-19, December 1995）に述べられている。前記ＣＰＵ１１８は、３２ビットデータバス１２２を経由してＦＰＵ１１４に結合されている。整数の累積乗算（積和演算）ユニット（ＩＭＡＣ）１２０がデータバス１２２に結合されている。図１の回路間のインタフェース信号は、ＣＰＵ１１８とＦＰＵ１１４との間のインタフェース信号１２４，１２５以外は図示を省略してある。前記ＣＰＵ１１８は、３２ビット論理アドレスバス１２６を経由してメモリマネージメントユニット装置（ＴＬＢコントローラ）１３４に結合されている。ＴＬＢは先見型アドレス

変換バッファを表している。ＴＬＢコントローラ１３４は命令とデータの混在型のＴＬＢ１３８を制御する。そのＴＬＢ１３８は３２ビット物理アドレスバス１３０を介してキャッシュコントローラ１４２に接続されている。前記キャッシュコントローラ１４２は命令とデータを混在させたキャッシュメモリ１４６を制御する。キャッシュメモリ１４６は３２ビットデータバス１２２を介してＣＰＵ１１８とＦＰＵ１１４に結合される。

【００２６】マイクロプロセッサ１１０は更に、周辺データバス１５０及び周辺アドレスバス１５１に接続されたシリアルコミュニケーションインタフェース１５２、タイマ１５３等を備え、それら周辺データバス１５０及び周辺アドレスバス１５１はバスステートコントローラ１５４を介して前記データバス１２２及び物理アドレスバス１３０にインタフェースされる。内外からの割込み要求の調停等は割り込みコントローラ１５５が行う。チップ外部とのバスインタフェースは外部バスインタフェース回路１５６によって行われる。この説明と図１に示された名称に基づけば、図１に示されているその他の回路の機能は当業者にとっては自ずと理解されるであろう。

【００２７】プロセッサ１１０の機能は以下の説明によって理解されるであろう。ＦＰＵ１１４は浮動小数点処理のためにデータや命令を必要とする。この例において、ＦＰＵ１１４は、キャッシュメモリにデータを格納し或いはデータを獲得するためのメモリのアドレシング能力を持ち合わせていない。これは、ＦＰＵのメモリアドレシング回路の必要性を取り除いてチップ面積を節約する。その代わりに、ＣＰＵ１１８がＦＰＵ１１４に代わってキャッシュメモリをアドレシングする。ＣＰＵ１１８はＦＰＵ１１４のためにメモリからデータのフェッチを行なうだけでなく、ＦＰＵ１１４のための浮動小数点命令を含む全ての命令をメモリからフェッチする。更にまた、以下により詳細を説明するように、ＦＰＵ１１４とＣＰＵ１１８との間でのデータ転送は、メモリアクセスを伴わずに、代わりにＦＰＵＬレジスタのようなレジスタを介して高速に行われる。

【００２８】データ又は命令を獲得するために、ＣＰＵ１１８は、論理アドレス（仮想アドレス）を計算し、３２ビット論理アドレスバス１２６を介してメモリマネージメント装置１３４にその論理アドレスを送ってメモリ１１６からのデータ又は命令を要求する。

【００２９】もしも、対応する物理アドレスが既にＴＬＢ１３８に格納されていないなら、ＴＬＢミスが発生し、ＴＬＢコントローラ１３４は、追加のマッピング情報を利用して論理アドレスを物理アドレスに変換するためのプログラムシーケンスを始める。メモリマネージメント装置１３４は、そのときＴＬＢ１３８に物理アドレスを格納する。ＣＰＵ１１８が再び同じアドレス範囲のデータを要求するかもしれないので、ＴＬＢ１３４は将来の利用のためにそのアドレスを新たなエントリとして格納する。ＴＬＢ１３８は３２ビット物理アドレスバス１３０を介してキャッシュコントローラ１４２に物理アドレスを送る。キャッシュコントローラ１４２は３２ビットデータバス１２２にデータ又は命令を置くように、データと命令が混在されるキャッシュメモリ１４６に指示する。仮に、要求されたアドレスがキャッシュメモリ１４６上で利用可能でないなら、キャッシュミスが発生し、ＣＰＵ１１８とＦＰＵ１１４の処理は、要求された情報が外部メモリからキャッシュメモリにフェッチされるまで、キャッシュミス信号を適用することによってフリーズされる。ここで、フリーズとは、命令それ事態の配列に起因して生ずるようなメモリコンフリクトやレジスタコンフリクトのような状態に応じてパイプラインを止めるストールとは異なる概念を一般に持つ。すなわち、フリーズとは、命令の配列（命令の実行順序）に起因しない要因、例えば、キャッシュミスやＴＬＢミスなどによってパイプラインをある期間止めることを意味する。ストールもフリーズも、ハザード（障害）に対処する一つの手段であることには変わりない。

【００３０】命令はＦＰＵ１１４と同様ＣＰＵ１１８によりデコーディングのために拾い上げられる。データはＣＰＵ１１８とＦＰＵ１１４によって共有される共通３２ビットデータバス上で利用可能である。ＦＰＵ１１４はデータ及び命令フェッチのためのメモリアドレッシングを行なう能力を持っていない。

【００３１】ＣＰＵ１１８とＦＰＵ１１４との間でのデータ交換は、専用のコミュニケーションレジスタＦＰＵＬを介して行われる。上述のように、その他のＲＩＳＣプロセッサにおいて、ＣＰＵ１１８とＦＰＵ１１４との間のデータ交換はキャッシュメモリ１４６のような転送メモリ（transfer through memory）を介して起こる。ＦＰＵＬレジスタのようなレジスタを介する転送はキャッシュメモリ１４６を介する転送に比べて速い。

【００３２】浮動小数点命令及び浮動小数点命令に関連するＣＰＵ命令のリストは図２０に示されている。これら命令は図２１～図５２に詳細が記述され、命令のＣ言語記述を含んでいる。図２１～図５２に列挙されたそれら命令には、命令の意味に関係する重要性を持っていないアンダーラインが示されている。

【００３３】図２にはＦＰＵ１１４とＣＰＵ１１８が命令を実行するのに利用するパイプラインを例示する。ＦＰＵパイプライン２１０とCPUパイプライン２１４は、単一の命令フェッチステージ２１８を共有する。更にＦＰＵパイプライン２１０は、デコードステージ（Ｄｆ）２２２、第１実行ステージ（Ｅ１）２２６，第２実行ステージ（Ｅ２）２３０、及びライトバックステージ（Ｓｆ）２１０の４ステージを更に持つ。ＣＰＵパイプライン２１４は、デコードステージ（Ｄｉ）２３８、実行ステージ（Ｅ）２４２、メモリアクセスステージ（Ａ２４

６）及びライトバックステージ（Ｓｉ）２５０の、更に４ステージを持つ。Ｄｉステージ２３８は、図２の矢印２５５によって示されるようなＦＰＵパイプラインステージに対してバイパス、ストール及びキャンセルのための信号を生成する。Ｄｉステージ２３８は、信号パス２５５を介して、Ｅ１，Ｅ２ステージ２２６，２３０にバイパス信号を供給し、全てのＦＰＵステージ２２２，２２６，２３０，２３４と共有Ｉステージ２１８にキャンセル信号を供給し、ストールに関連する信号をＤｆステージに供給し、そしてストール信号をＥ１ステージに供給する。これらの信号は以下で更に説明する。

【００３４】データバス１２２の上で利用出来る命令は、インストラクションフェッチステージ２１８によって最初獲得される。双方のデコードステージＤｆ２２２及びＤｉ２３８は、インストラクションフェッチステージでフェッチされた命令をデコードする。デコードステージの第１フェーズは、ＣＰＵ命令かＦＰＵ命令かを識別することを含む。ＦＰＵ命令は命令の上位４ビットにおけるＦ（１６進数）によって識別される。命令が浮動小数点タイプでないなら、Ｄｆステージ２２２はこれ以上命令をデコードしない。同様に、Ｄｉステージ２３８は浮動小数点命令を完全にデコードしない。Ｄｉステージ２３８は、実行されるべき浮動小数点機能を識別するために浮動小数点命令をデコードすることはしない。これはハードウェアの複雑性に対して縮小をもたらし、重要なことである。もしも単一のデコードステージだけが利用されるなら、ＦＰＵのデータパスを制御するために要求される信号は、ＣＰＵ１１８からＦＰＵ１１４を交差しなければならず、チップ面積を増加させることになる。フェッチされた命令が浮動小数点命令であるとき、ＦＰＵパイプライン２１０のＥ１ステージ２２６は、命令の実行を始める。ＦＰＵパイプライン２１０のＥ２ステージ２３０は、その浮動小数点命令の実行を完了する。命令の必要条件に依存して、ＦＰＵパイプライン２１０のＳｆステージは、浮動小数点レジスタに命令の実行結果をストアすることができる。

【００３５】フェッチされた命令が、整数を操作するための命令のような、ＣＰＵ命令である場合、ＣＰＵパイプライン２１４のＥステージ２４２は、当該命令を実行する。ＣＰＵパイプライン２１４のＡステージ２４６は、実行されている特定の命令によってそれが要求されたとき、キャッシュメモリ１４６をアクセスする。最後に、ＣＰＵパイプライン２１４のＳｉステージ２５０は、命令実行結果を、例えばＣＰＵレジスタ４１０の一つに書き込むことができる。有利なことに、ＦＰＵパイプライン２１０又はＣＰＵパイプライン２１４の一つだけを利用することを要求する命令は、利用されないパイプラインに関しては単にそれを通過する命令、ということになる。例えば、命令フェッチステージ２１８が整数加算命令をフェッチしたとき、ＣＰＵパイプライン２１４は実行ステージ２４２で整数加算を実行し、Ｓｉステージ２５０でレジスタにその実行結果を格納する。しかし、整数加算命令がデコードされると、ＦＰＵパイプライン２１０のＤｆステージ２２２は整数加算命令をＥ１ステージ２２６に通過させる。整数加算命令は後の複数クロックサイクルの間、ＦＰＵパイプライン２１０の残りのステージを通過し続ける。同様に、フェッチされた命令が純粋な浮動小数点命令であるとき、前記Ｄｉステージ２３８は浮動小数点命令をＥステージ２４２に通過させ、そして、後の複数サイクルの間、ＣＰＵパイプライン２１４の残りのステージを単に通過させる。ここで上記通過とは、表裏の関係にあるＣＰＵパイプラインとＦＰＵパイプラインにおいて実質的なオペレーションを伴わないパイプラインステージがそのパイプラインを単に通過していくということを意味する。

【００３６】ここで、ＣＰＵパイプラインとＦＰＵパイプラインとを図２に例示されるように構成することの意義を説明する。すなわち、ＦＰＵパイプライン２１０とＣＰＵパイプライン２１４のパイプライン段数（パイプラインのステージ数）を相互に等しくしてある。これによって、ＣＰＵパイプライン２１４とＦＰＵパイプライン２１０とのフロー制御等のためのパイプライン制御用のリソースがＦＰＵパイプラインとＣＰＵパイプラインで殆ど共有出来るようになる。このパイプラインリソースの共有によってパイプライン制御のための物理的な回路規模を縮小することができる。例えば、図１８に示されるようなレジスタコンフリクトを生ずる場合を一例とすると、先の命令“Ａｄｄ　Ｒ０，Ｒ１２”のディスティネーションレジスタ（Ｒ１）が後の命令“Ａｄｄ　Ｒ１，Ｒ２”のソースレジスタ（Ｒ１）である場合、先の命令のディスティネーションレジスタ（Ｒ１）に演算結果が得られた後でなければ、後の命令にとってそのレジスタ（Ｒ１）の値を参照することは無意味である。そのために、図示はしないが、パイプラインＰ１のライトバックＳｉステージでレジスタＲ１に演算結果が得られるまで、パイプラインＰ２をストールすることができる。このとき、パイプラインＰ１の実行ステージＥで演算された演算結果をパイプラインＰ２の実行ステージＥにバイパスして渡すことにより、ストールを行わずに済む。同様に、パイプラインＰ３～Ｐ５で示されるように、レジスタＲ１１に関してデータコンフリクトを生ずる命令“Ａｄｄ　Ｒ１０，Ｒ１１”と“Ａｄｄ　Ｒ１１，Ｒ１２”との間に別の命令が介在されている場合には、パイプラインＰ３の実行ステージＥで演算された演算結果をメモリアクセスステージＡでパイプラインＰ５の実行ステージＥにバイパスして渡すことにより、ストールを行わずに済む。上述のストールを行う期間、そして、上記バイパスをどのステージからどのステージに対して行うかという制御は、パイプラインステージの長さ（段数）に依存することが理解されるであろう。通常、ストール

の制御はストールされるべきパイプライン側で制御し、また、バイパス制御はバイパスされるデータの受け側で制御する。そうすると、ＦＰＵパイプラインとＣＰＵパイプラインの段数が相違する場合には、バイパスに代表されるようなパイプライン間でのデータ交換、ストール、フリーズ、及びキャンセル等のための制御を行うリソース、即ち、パイプラインのフロー制御などを行うためのロジック回路やバイパスのための制御線などを、ＣＰＵパイプラインとＦＰＵパイプラインとの間で殆ど個別化しなければならなくなる。この実施の形態に示される例では、図２に例示されるように、ＦＰＵパイプライン２１０とＣＰＵパイプライン２１４のパイプライン段数を相互に等しくし、ＣＰＵパイプライン２１４とＦＰＵパイプライン２１０とのフロー制御等のためのパイプライン制御用のリソースをＦＰＵパイプラインとＣＰＵパイプラインで殆ど共有させることができる。図１９はＣＰＵパイプラインとＦＰＵパイプラインを制御するリソース、特にフロー処理のための論理の共有状態を概念的に示すものであり、ＣＰＵ命令のためのフロー処理の論理がＦＰＵ命令のためのフロー処理に共有され、そのフロー処理の論理にはＦＰＵ命令のための付加ロジックは僅かに追加されるだけである。この付加ロジックはフロー制御の対象がＦＰＵパイプラインかＣＰＵパイプラインかを示すフラグなどとされる。

【００３７】以下、ＣＰＵパイプラインとＦＰＵパイプラインの種々の同期化の具体例を説明する。

【００３８】ある命令はＦＰＵパイプライン２１０とＣＰＵパイプライン２１４との間でのデータ転送を要求する。そのような命令の一例として、ＣＰＵによるコミュニケーションレジスタ（ＦＰＵＬ）へのロード命令（ＬＤＳＲｍ，ＦＰＵＬ）、すなわち、ＣＰＵロード・コミュニケーションレジスタ命令がある。この命令は、ＦＰＵパイプライン２１０とＣＰＵパイプライン２１４並びにそれらパイプライン２１０，２１４の間でのデータ転送タイミングを示す図３で示されている。パイプラインの構成は図２で説明した通りでありる。パイプラインの夫々のステージはまた、例えば一つのクロック３０８の位相（明らかなように二つのクロックの位相は示されていない）の単一のクロックサイクルに同期することに注意されたい。すなわち、一つのパイプラインステージはクロック３０８の１サイクルで実行される。ＦＰＵＬはＦＰＵ１１４に含まれるレジスタである。このレジスタＦＰＵＬはデータ処理上、ＦＰＵ１１０とＣＰＵ１１８が共有する。ＣＰＵによるコミュニケーションレジスタ（ＦＰＵＬ）へのロード命令はＣＰＵ命令である。しかしながら、上述のように、全ての命令はＦＰＵ及びＣＰＵパイプライン２１０，２１４における双方のデコードステージ２２２，２３８でデコードされる。だから、ＣＰＵによるコミュニケーションレジスタ（ＦＰＵＬ）へのロード命令をデコードするや否や、Ｄｆステージ２２２は、ＦＰＵ１１４がレジスタＦＰＵＬへのアクセスを制御することになるので、ＦＰＵパイプライン２１０を実質的に動作させるべきであると認識する。最初、ＣＰＵがコミュニケーションレジスタ（ＦＰＵＬ）にロードする命令は、ＣＰＵパイプライン２１４のＥステージによって実行される。同時に、ＦＰＵパイプライン２１０のＥ１ステージ２２６は、文字（Ｔ）３１０によって示されるように、何ら動作することなく命令を通過させる。言い換えれば、ＣＰＵがコミュニケーションレジスタ（ＦＰＵＬ）にロードする命令は、Ｅ１ステージに対して単に通過することになる。

【００３９】ここで、Ｅ１ステージに関連した別の説明を行う。パイプライン２１０，２１４の夫々のステージは、実行に１サイクルを要する。しかしながら、命令がパイプラインステージに１サイクルよりも多くの時間を費やすという、特別な場合がある。その場合に、命令はその特定のパイプラインステージを繰り返すことになる。例えば、図２８及び図２９に示されるような浮動小数点除算命令（ＦＤＩＶ）は、１３サイクルのレーテンシ（待ち時間）を持つ。ここで、レーテンシは、命令が例えばＦＰＵパイプライン２１０の実行ステージ２２６，２３０で費やすサイクル数の合計の尺度である。浮動小数点命令はＥ２ステージ２３０で１サイクルを費やす。それによって明らかなように、浮動小数点除算命令はＥ１ステージで１２サイクルを費やす。図２８には浮動小数点除算命令のピッチ（Pitch）が示され、それは一般に、現在の命令の後に続く命令がパイプラインで実行開始できる前のクロックサイクルの尺度である。例えば、浮動小数点除算命令のピッチは１２サイクルに等しいから、浮動小数点除算命令の後に続く次の命令は１２サイクルの後に実行することができる。ピッチの値１２は、浮動小数点除算命令はＥ１ステージ２２６で１２サイクル費やすということを示す。したがって、次の浮動小数点命令はＥ１ステージ２２６に入る前に、１２クロックサイクル待たなければならない。

【００４０】図３に示されるコミュニケーションレジスタ（ＦＰＵＬ）へのロード命令の例に戻ると、同じ命令がＦＰＵ及びＣＰＵパイプライン２１０、２１４の双方に流れるので、Ｅ１ステージ２２６のリソースは、ＣＰＵパイプライン２１４のＥステージが実行するサイクル数と同じだけ保持される。次に、ＣＰＵパイプライン２１４のＡステージ２４６が、データバス１２２に、ＬＤＳ命令（ＣＰＵロード・コミュニケーションレジスタ命令）で参照される“Ｒｍ”というレジスタの内容を載せる。この命令はキャッシュメモリのアクセスを要求する命令ではないから、Ａステージ２４６はＣＰＵレジスタファイル４１０のレジスタＲｍのデータをデータバスにロードするだけである。すなわち、ステージＡ（Ｔ’）３１４はメモリアクセスが起きないことを表している。

【００４１】ＣＰＵパイプラインがデータバス１２２に

レジスタ“Ｒｍ”の内容を置いているとき、ＦＰＵパイプライン２１０のＥ２ステージ２３０は、文字“Ｔ”によって示されるように、どんな動作も伴わずに命令を通過させる。ＣＰＵパイプラインのＡステージ２４６は、データーレディー時間３１８の限られた期間、レジスタＲｍの内容をデータバス１２２上で利用出来るようにする。データーレディー時間とは、ＣＰＵロード・コミュニケーションレジスタ命令に関連するデータ転送によってデータバスがビジーである時間とされる。データバス１２２上のデータが利用可能な前記期間３１８の間、ＦＰＵパイプライン２１０のライトバックステージＳｆ２３４がデータバス１２２上のデータを獲得し、そのデータをレジスタＦＰＵＬにストアする。

【００４２】この例からも明らかなように、ＣＰＵパイプライン２１４とＦＰＵパイプライン２１０とのパイプラインステージの段数は同じである。したがって、ＣＰＵ１１８にとってはメモリへの書込みと同様のシーケンスを採れば（実際にはメモリアクセスは行われない）、ＦＰＵパイプライン２１０はステージＳｆでそのデータをＦＰＵＬレジスタに取り込むことができる。

【００４３】ＣＰＵストア命令“STS ＦＰＵＬ，Ｒｎ”は、図４に示されるように、前記同様、二つのパイプライン２１０、２１４によって実行される。そのＣＰＵストア命令は、レジスタ（ＦＰＵＬ）の内容をＣＰＵ汎用レジスタＲｎにコピーする。このＣＰＵストア命令の場合、ＦＰＵ１１４は、データバス１２２上でレジスタＦＰＵＬの内容を利用可能にする期間３２２を制御する。すなわちＦＰＵパイプラインはＦＰＵＬレジスタの内容を３２２の期間だけデータバスに載せ、ＣＰＵパイプライン２１４はこのデータをステージＳｉでレジスタＲｎに取り込む。

【００４４】図５はＦＰＵパイプライン２１０の更に詳細な回路４０６を示す。図５の回路には、夫々ＦＰＵパイプラインステージ２２２，２２６，２３０，２３４に含まれるマスタ・スレーブ形式のラッチ４１４，４１８，４２２，４２６が示される。ＦＰＵパイプラインステージ２２２，２２６，２３０，２３４はそれらの出力をそれらに対応されるラッチ４１４，４１８，４２２，４２６に、１相目のクロック信号４０８の立ち下がりエッジ又は２相目のクロック４１０の立ち下がりエッジの何れかに同期してストアする。また、以下に記述されるバイパス信号４２８が示されている。ＦＰＵパイプライン回路４０６の機能は、すぐにそして以下で述べられる例によって示される。ＣＰＵロード・コミュニケーションレジスタ命令の例において、ＦＰＵデコードステージ（Ｄｆ）２２２は、ＦＰＵＬレジスタにデータがロードされるように、マルチプレクサ４３４の選択信号経路４３０上で選択信号をアサートすることによってデータバス１２２からレジスタＲｍの内容を獲得する制御を行なう。

【００４５】図６はＣＰＵ命令とＦＰＵ命令の二つの命令の実行手順におけるパイプライン２１０と２１４との同期の一例を示す。特に図６は、前述のＣＰＵストア命令が後に続くことになる浮動小数点切り捨て及び整数へ経の変換命令５１０（図４１及び図４２参照）とを示す。図６において、一つの命令におけるＦＰＵパイプライン２１０とＣＰＵパイプライン２１４の夫々のパイプラインステージは、図示を簡単にするために一つの連続するブロックに統合して図示されている。だから、例えば、二つのデコードステージを示す代わりに、Ｄｆ２２２とＤｉ２３８を示す文字“Ｄ”を伴って示されている。図６の表示において時間は、クロック信号５１４（簡単のため、クロック信号の二つの相は示されていない）の一つの相によって示されるように、左から右に経過する。このパイプラインの表現形式は、適当なステージを単に命令が通過する状態を他と区別して表すために、パイプラインステージの指定において、文字“Ｔ”が括弧の中に挿入されている、という点を除いて、この技術分野では標準的である。例えば、浮動小数点ユニットのＦＰＵパイプライン２１０の実行ステージ（Ｅ１）２２６が浮動小数点切り捨て命令５１０を実行しているとき、ＣＰＵパイプライン２１４の実行ステージ（Ｅ）２４２は、文字“Ｔ”によって示されるように何ら動作することなく単に命令を通過させる。

【００４６】図６のＦＰＵ命令“ＦＴＲＣ　Ｆｒｍ，ＦＰＵＬ”とＣＰＵ命令“ＳＴＳ　ＦＰＵＬ，Ｒｎ”とを比べれば明らかなように、ＦＰＵＬに関してデータコンフリクト（レジスタコンフリクト）を生ずる。データコンフリクトによるストールの発生を防止するために、即ち、ＣＰＵパイプライン２１４がストア命令を実行しているとき、ＣＰＵパイプライン２１４でのストールを防止するために、前記命令“ＦＴＲＣ”におけるＣＰＵデコードステージ２３８はバイパス信号経路２５５にバイパス信号５２２をアサートする。バイパス信号５２２のアサートは、命令“ＦＴＲＣ”のＥ２ステージ２３０が浮動小数点切り捨て命令５１０を実行完了した後であって命令“ＳＴＳ”のＥ２ステージがストア命令を実行する準備ができたとき、当該ストア命令“ＳＴＳ”のＥ２ステージ２３０の入力に、前記命令“ＦＴＲＣ”にけるＦＰＵパイプライン２１０のＥ２ステージ２３０の出力を利用可能にさせる。

【００４７】図５のパイプライン回路はデータの上記バイパスを達成するようになっている。すなわち、Ｅ２ステージのラッチ４２２の出力はバイパス経路４３８で利用可能になる。バイパス経路４３８は、マルチプレクサ４４２の入力である。ＣＰＵデコードステージ２３８からのバイパス信号４２８はＥ２ステージのラッチ４２２の出力を選択し、それによって、それがＥ２ステージ２３０を通してリサイクルされることができる。浮動小数点切り捨て命令５１０の結果がＥ２ステージ２３０の出

力ラッチ４２２を介して利用可能になる時間は、図６において５２６で示される。ＣＰＵのＤｉステージ２３８からのバイパス信号４２８は、マルチプレクサ４２２を通してデータの転送を可能にする適切な時間にアクティブになる。既に説明したように、上記ストア命令“ＳＴＳ”においてＦＰＵ１１４は、ＣＰＵレジスタＲｎに転送するためにデータバス１２２上のデータが利用可能であるところの期間３２２を制御する。

【００４８】主に、ＦＰＵパイプライン２１０とＣＰＵパイプライン２１４の同期化は、双方のパイプラインでストールとフリーズの効果をもたらすこれらパイプライン２１０，２１４の何れか一方がストールとフリーズの制御機能を持つことによって達成される。図２に示されるように、ＣＰＵパイプラインのデコードステージ（Ｄｉ）２３８は、矢印２５５によって示されるストール信号データ経路を介してＦＰＵパイプラインをストールする。パイプラインストールは、命令シーケンスの間、例えば、最初の命令がレジスタにライトするとき、そのレジスタの内容が次の命令によって利用される間に、起こる。前記次の命令に、更新されていないレジスタの内容を利用させないようにするために、前記最初の命令によってレジスタが更新されるまで、当該次の命令がストールされる。ここで説明する実施の態様では、そのようなストールは、一つの命令の実行ステージ（Ｅ）２４２の出力を次の命令の実行ステージ（Ｅ）２４２の入力にバイパスすることによって発生されれないようになっている。類似のバイパスはＦＰＵパイプに存在する。ストールの間、ストールされるべきパイプラインステージ（例えばＥステージ２４２又はＥ１ステージ２２６）にはＮＯＰ（ノン・オペレーション）が生成される。

【００４９】このように、レジスタコンフリクトを生ずるとき、ストールを発生させないようにするにはバイパスを利用することができる。このとき、バイパスの制御は、図２からも明らかなように、ＣＰＵデコードステージＤｉが行う。すなわち、ＦＰＵパイプラインとＣＰＵパイプラインとのパイプラインステージの段数が基本的に同一にされるから、ストール、フリーズ、バイパスを制御するフロー制御の論理はＣＰＵパイプラインとＦＰＵパイプラインとの間で共有することができる。この意味において、ＦＰＵパイプラインとＣＰＵパイプラインのフロー制御のためのリソースは、双方のパイプラインで共有することができる。パイプライン制御のためのリソースを双方のパイプライン２１０、２１４で共有出来るが故に、フロー制御のハードウェアをＦＰＵとＣＰＵが別々に持たずに済み、ＣＰＵパイプラインのフロー制御ロジックをＦＰＵパイプラインのフロー制御にも流用することができ、これにより、ＣＰＵパイプラインとＦＰＵパイプラインとを同期化することが容易になる。

【００５０】次に、追加のストール条件に関連する回路を例示する。図７はストールの第１のタイプすなわちロード・ユース・ストールを例示するパイプラインダイヤグラムである。図７には３個の命令６１０，６１４，６１８のシーケンスが存在する。第１の命令６１０は第２の命令６１４と同様にＣＰＵ命令である。第３番目にフェッチされる命令は浮動小数点命令である。命令６１０は、レジスタＲ２に格納されているアドレスでメモリ１４６の内容をレジスタＲ１にロードするロード命令である。第２の命令６１４はレジスタＲ１の内容をレジスタＲ２の内容に加算する命令である。第１の命令６１０はＣＰＵパイプライン１１４のＡステージ１４６でメモリ１４６をアクセスしており、第２の命令６１４はストールされる。さもなければ、第２の命令６１４は、命令６１０がレジスタＲ１の内容を更新している間に、レジスタＲ１の更新されていない内容をアクセスするであろう。ストールされると、命令６１４はデコードステージＤから処理をやり直す。

【００５１】一つのパイプラインにおけるストールは、両方のパイプライン（ＣＰＵパイプライン及びＦＰＵパイプライン）でストールを引き起こすので、浮動小数点命令である第３の命令６１８は、第２の命令６１４とともにストールされることになる。ＣＰＵパイプライン２１４のＤステージ２３８は、双方のパイプライン２１０、２１４をストールするためのストール信号６２２を生成する。追加のストールを防止するために、Ａステージ２４６の内容はＣＰＵパイプライン２１４のＥステージ６４２にバイパスされている。このバイパスが行われないなら、命令６１０のライトバックステージＳが完了されるまで、ストールされなければならない。

【００５２】図８はストールのもう一つのタイプ、すなわちメモリ・アクセス・コンフリクト・ストールを例示する。図８に示されるストールは、二つの命令が同時にキャッシュメモリ１４６をアクセスしようとすることによって起こる。図８は４個の命令のシーケンスを実行するためのパイプライン・ダイヤグラムが示される。第１の命令は図７のＣＰＵムーブ命令６１０としてのＣＰＵムーブ命令７５１である。図７の命令に対し、命令７１５はメモリ１４６をアクセスする。

【００５３】図８において、第１の命令７１５は、命令フェッチステージ２１８により単一フェッチ動作でフェッチされた二つの命令の内の一つであり、図８では簡単のために他方の命令フェッチは図示されていない。すなわち、１６ビット固定長命令に対する命令フェッチは３２ビットデータバスを介して２命令単位（３２ビット）で行うことができるようにされているからである。図８において、命令フェッチステージ２１８がメモリ１４６から第４番目の命令をフェッチする準備が完了するであろう時、第１の命令７１５はＡステージ２４６でメモリ１４６をアクセスするから、Ｄｉステージ２３８は双方のパイプ２１０，２１４をストールする。したがって、第４番目の命令７２０は、ストールされない場合に比べ

て１クロックサイクル遅れてフェッチされる。

【００５４】更に、第３番目の命令７３０は、更に以下で記述される図１１の説明に含まれるストール信号１０１０の適用によって、Ｄｉステージがやり直しのために繰り返される。更に、ＮＯＰ（ノン・オペレーション）が、ストール信号７２５の適用によって、第３の命令７３０のＥステージ６４２に挿入される。そして、一方をストールすることは同様に他方をストールすることになるから、双方のパイプライン２１０，２１４がストールされる。

【００５５】図９はストール信号６２２，７２５を生成する（ＣＰＵパイプライン２１４のＤｉステージ２３８における）回路８１０のダイヤグラムである。図７のストール信号６２２を生成するために、回路８１０は、ディスティネーションレジスタ（図７の命令６１０のＲ１）がソースレジスタ（図７の命令６１４のＲ１）と同一かどうかを決定するためにコンパレータ８１５を利用する。もしもそのような同一性があるなら、ＯＲ回路８２０がストール信号６２２を生成する。

【００５６】同様に、メモリアクセス回路８２５がメモリアクセスのコンフリクトを決定するなら、図８について上記したように、そのとき、ＯＲ回路８２０はストール信号７２５を生成する。

【００５７】図１０はＣＰＵデコードステージ２３８が、ＣＰＵ及びＦＰＵパイプライン２１０，２１４の双方を同時にストールすることによりどうやって同期化を維持するかについて例示する。図１０において、２個の命令は連続的に実行される。最初、図３で説明したＣＰＵロード・コミュニケーションレジスタ命令がＦＰＵ及びＣＰＵパイプライン２１０，２１４によってフェッチされて実行される。次に、命令フェッチステージ２１８が浮動小数点命令９１０をフェッチする。（命令フェッチステージ（Ｉ）２１８は３２ビットの命令を２個一度にフェッチする。このフェッチは偶数ワード境界で生ずる。だから、命令毎にフェッチサイクルを始める必要はない。）浮動小数点命令９１０はレジスタＦＰＵＬの内容を整数値として解釈する。その浮動小数点命令９１０は更にその整数値を浮動小数点数に変換する。最後に、浮動小数点命令９１０は浮動小数点数を浮動小数点レジスタＦＲｎに格納する。

【００５８】図１０において、浮動小数点命令９１０は、双方の実行ステージ（Ｅ１）２２６及び（Ｅ）２４２にストール信号９１４を用いるＣＰＵ１１８のデコードステージ（Ｄｉ）２３８の結果として、ＦＰＵ及びＣＰＵパイプライン２１０，２１４をストールさせる。すなわち、双方のパイプライン２１０，２１４の実行ステージ２２６，２４２がストールされる。そのようなストールは文字“Ｘ”９１８によって共通に表される。このストールはここでは適切であり、それは、浮動小数点命令９１０が普通に実行準備完了されるであろうとき、命令“ＬＤＳ　Ｒｍ，ＦＰＵＬ”を実行するＣＰＵ１１８のＡステージ２４６はデータバス１２２上のデータを未だ利用可能にしていないからである。１サイクルストールされることにより、浮動小数点命令９１０はストール信号９１４がローレベルにされる途端に処理を継続する。命令“ＬＤＳ　Ｒｍ，ＦＰＵＬ”を実行するＣＰＵパイプライン２１４のＡステージ２４６の出力において利用可能なデータはバイパス経路９２２を経由してデータバス１２２にバイパスされ、そこから命令９１０を実行するＦＰＵパイプライン２１０のＥ１ステージ２２６の入力にバイパスされる。このバイパス９２２は更にストールを追加することを防止する。なぜならば、浮動小数点命令９１０は、データバス１２２上で利用可能なＦＰＵＬレジスタの値を処理することによって命令を実行することができるからである。

【００５９】同様に、この実施の態様において、パイプライン・フリーズが一つのパイプラインで起きるとき、それは双方のパイプライン（ＦＰＵ及びＣＰＵパイプライン２１０，２１４）のフリーズを同時に引き起こす。パイプラインフリーズの間、パイプライン２１０及び２１４における全ての動作は停止する。双方のパイプライン２１０，２１４のフリーズは、例えばキャッシュミスの結果として生ずる。キャッシュミスは、例えば、キャッシュメモリ１４６に存在しないデータをＣＰＵ１１８が要求したとき生ずる。その場合、キャッシュコントローラ１４２はＣＰＵ１１８にキャッシュミスが生じたことを示す信号を送る。また、ＣＰＵ命令が結果を要求する乗算処理をＩＭＡＣ１２０が完了していないとき、そのＩＭＡＣ１２０はＦＰＵ１１４と同様にＣＰＵ１１８にもビジー信号を送り、それによってパイプラインフリーズを生ずる。

【００６０】図１１はＦＰＵパイプライン２１０のデコードステージＤｆ２２２の回路ダイヤグラムである。デコードステージ（Ｄｆ）２２２の機能はＣＰＵ１１８のデコードステージ（Ｄｉ）２３８によって生成されるストール信号１１１０（図１１参照）に依存する。このストール信号１１１０は、ＦＰＵパイプライン２１０の実行ステージ（Ｅ１）２２６で利用され、Ｅ１ステージ２２６でＮＯＰを生成する。Ｄｆステージ２２２がストールの条件を検出すると、図１１のＤｆステージ２２２はＤｆの出力１０１４からのＤｆ２２２でデコードされた命令を帰還する。経路１０１８を介する帰還は、マルチプレクサ１００８の入力１００６（セレクタ＝１）を選択するためのＤｉステージ２３８からの制御信号１０１０によって達成される。

【００６１】図１１はまた、フリーズ信号選択経路１０２２を示す。上述のように、フリーズ信号は全部のパイプラインで全ての処理を停止させる。同様に、フリーズ信号は、上述したように、双方のパイプライン２１０，２１４に適用される。フリーズ信号１０２２はラッチ１

０２４をディスエーブルにする。また、図１１にはキャンセル信号選択経路１０２６が示されている。キャンセル信号選択経路１０２６でキャンセル信号を選択することは、マルチプレクサ（ＭＵＸ）がＮＯＰ１０３０を挿入することであり、これにより、パイプラインのそのポイント（ステージ）でどの命令もキャンセルできる。１０３４で示される部分は更にＦＰＵパイプライン２１０のデコードステージ２２２に適用される信号を示す。

【００６２】１０３４で示される部分の記述において、シンボルで示される信号（例えば信号Ａ，信号Ｂ）の説明はＣ言語標記である。１０３４で示される部分において、信号名の定義における垂直線は論理的な“ＯＲ”（論理和）を示す。“＆”は論理的な“ＡＮＤ”（論理積）を示す。“～”は論理的な反転を示す。

【００６３】図１２はＦＰＵパイプライン２１０の第１の実行ステージ（Ｅ１）２２６のダイヤグラムを示す。ストールが第１の実行ステージ２２６で適切であるとき、上述のように、ＣＰＵ１１８のデコードステージ２３８はストール信号パス１１１０にストール信号を適用する。これはＦＰＵデコードステージ２２２の出力１０３８がＥ１ステージに渡されるのを妨げる。代わりに、経路１１１０でストール信号をアサートすることにより、Ｅ１ステージ２２６に挿入されるべきＮＯＰ１１１４が生成される。図１２におけるフリーズ及びキャンセル信号の応用は、図１１のそれと同様である。それらの信号については１１１４で示される部分に更に詳述される。

【００６４】図１３はＦＰＵパイプライン２１０のステージ（Ｄｆ）２２２，（Ｅ１）２２６，（Ｅ２）２３０，及び（Ｓｆ）２３４に関する詳細な回路ダイヤグラムである。ＦＰＵパイプライン回路１２０８は幾つかの入力を示す。入力１２１０は第１オペランドのためであり、入力１２１４はＦＰＵ又はＣＰＵ命令の第２オペランドのためである。入力１２１８は、Ｅ２ステージ２３０の出力をＥ１ステージの入力に戻すためのバイパスデータを受け入れる。入力１２２２はデータバス（ＳＤ＿２）１２２の内容をＥ１ステージ２２６の入力にバイパスするためである。入力１２２６は浮動小数点レジスタファイルのＦＲ０レジスタから入力である。夫々部分的にハッチングが付されている複数のラッチは、クロック信号の２相目に同期して動作される。上述の説明と図１３に記述された名称に基づけば、当業者は回路１２０８の残りの部分を容易に理解することができるであろう。

【００６５】ＦＰＵとＣＰＵパイプライン２１０，２１４の間における同期化に利用される命令シーケンスの他の種類としては、ＣＰＵ分岐命令が後続される浮動小数点比較命令がある。図１４はそのようなシーケンスを示す。浮動小数点比較命令は、浮動小数点レジスタＦＲｍ１３１８の値が浮動小数点レジスタＦＲｎ１３２２の値に等しいとき、１の値をＴビットにセットする。Ｔビットの値は、分岐先へのジャンプが実行されるか否かを決定する。分岐命令１３１４はＣＰＵ命令である。Ｔビットが１なら、分岐命令は分岐先から新たな命令のフェッチを生じさせる（この例の場合にはキャッシュメモリ１４６からのフェッチとなる）。図１４において、Ｔビットの値はＦＰＵパイプライン２１０のＥ１ステージ２２６からＣＰＵパイプライン２１４のＥステージ２４２にバイパスされる。換言すれば、浮動小数点比較命令を実行しているとき、Ｔビットの出力はＥ１ステージで行われる。Ｔビットのそのような先方へのバイパスは、浮動小数点比較命令のの後に続くＣＰＵ分岐命令１３１４の実行におけるストールを防止する。ＴビットがＥ２ステージで出力されるなら、１サイクルのストールが必要になってしまうからである。

【００６６】図１５はＥ１ステージ２２６からＥステージ２４２へＴビットをバイパスするための回路を示す。さらに、図１５の回路はＣＰＵパイプラインのステージの間でＴビットをバイパスする能力を持つ。図１４のバイパスの一例に対応させて説明すると、最初の命令例えば浮動小数点比較命令１３１０がＴビット１４１０の値を１にセットしたとき、ＦＰＵは、Ｅ１ステージ２２６からＴビットをマルチプレクサ１４１８を介して選択するために、Ｔビット選択パス１４１４を活性化する。選択回路１４２２は、ＣＰＵパイプラインのＡステージラッチ１４２６若しくはＳｉステージラッチ１４３０のような幾つかの可能なＴビット・ソースから、又はステータスレジスタ１４３４からでえさえ、選択できるようになっている。選択信号１４３２は、デコードステージ（Ｄｉ）２３８から供給され、パイプラインステージで現在実行中の命令に基づく。分岐アドレス発生回路１４３８は、ＣＰＵが次の命令をフェッチすることができるアドレスを生成する。もちろん、Ｅ１ステージ２２６からのＴビット１４１０が０の値を持っていたなら、分岐アドレス発生回路１４３８は、プログラムの順番において分岐命令の後にすぐに続く命令のアドレスを選択することになる。次の命令１４４２のアドレスは、目的とする命令をフェッチするためにＣＰＵパイプライン２１４のＥステージ２４２に進む。

【００６７】命令は時々例外を引き起こす。例えば、命令が０で除算しようとしたり、不当命令コードを利用したりするときである。例外が生ずると、一般に例外ハンドラが例外を処理するための命令系列を実行する。その後、例外ハンドラは、ＣＰＵ１１８に、例外を引き起こした命令を伴うプログラムの実行を続けるのを許容する。すなわち、例外を生じた命令を再実行する。この実施の態様において、例外は、プログラムが規定する命令実行順序に対して実際の命令実行順序を乱さないという意味において、正確である。例えば、浮動小数点命令の例外はＦＰＵパイプライン２１０のＥ１ステージ２２６で検出される。ＣＰＵ１１８がコンピュータプログラム

のオリジナルプログラムの順番を維持するとき、例外は、例外であるにもかかわらず、正確である。オリジナルプログラムの順番は、コンパイルされたコンピュータプログラムとしての、オリジナルな命令シーケンスの順番である。換言すれば、例外が正確であるとき、コンピュータプログラムの命令は、それらが純粋に連続的なシステム、すなわちパイプライン能力を持っていないものによって実行されているかのように、実行される。例外ハンドラは、例外が起こる前のプロセッサ１１０の状態のような、装置の状態を退避する。

【００６８】双方のパイプライン２１０，２１４で共通のストールとフリーズを持つのと同様に、例外はＣＰＵ及びＦＰＵパイプライン２１０，２１４において双方同じパイプステージ数を持つこと（パイプライン２１０と２１４の同期化）によって正確にされる。双方のパイプライン２１０，２１４におけるストールとフリーズの同期化を達成するために、長い浮動小数点命令のための追加回路が含まれている。図１６は、浮動小数点命令が比較的長い時間を完了に費やすときでも、ビジー信号をアサートすることによって、浮動小数点パイプライン２１０のデコードステージ２２２が、例外を生じさせる場合にどうやって、（プログラムの記述によって規定される命令実行順序に対し）実際の命令の実行順序を正確に保つかを示す。ビジー信号１２４がアサート（例えばハイレベルにセット）されるとき、その他の浮動小数点命令は、Ｅ１ステージ２２６が自由にされるまでＦＰＵパイプライン２１０のＤｆステージ２２２を超えて進むことはできない。ビジー信号がアサートされる間、ＣＰＵ命令がフェッチされたときはＣＰＵパイプラインを用いてその命令を実行する。一方、ビジー信号がアサートされている間、他の浮動小数点命令がフェッチされるなら、当該命令はＤｆステージで繰り返される。これに応ずるＣＰＵパイプライン２１４もストールされる。

【００６９】図１６において、夫々の命令は１～６で示されるシーケンスにおいてＩステージ２１８によってフェッチされ、ＦＰＵパイプライン２１０又はＣＰＵパイプライン２１４によって実行される。図１６は６個の命令のシーケンスを例示する。第１の命令は浮動小数点除算命令１０１０である。ＦＰＵパイプライン２１０のデコードステージＤｆ２２２が一旦浮動小数点命令１５１０を解読すると、Ｄｆステージ２２２はビジー信号１５０８をアサートしてＣＰＵのデコーダに供給する。ビジー信号経路１２４にそのビジー信号１５０８がアサートされた後に続いて、２）、３）で示されるような、次のＣＰＵ命令が実行を続けることができる。しかしながら、４）で示される次のＦＰＵ命令、すなわち浮動小数点加算命令１５１４は、ＣＰＵパイプライン２１４を上述のように、ストール信号１１１０の適用によってストールさせようとする。ＦＰＵパイプラインステージ（Ｅ１）２２６、（Ｅ２）２３０、（Ｓｆ）２３４は、１）の命令の実行を続けるが、４）の命令に代表されるように、１）の命令の後の新たな浮動小数点命令は、Ｄｆステージから先に進むことは許されない。一旦第１の浮動小数点命令がＥ１ステージによって実行されるのが終了されると、ＦＰＵパイプライン２１０のＤｆステージ２２２はビジー信号１５０８をネゲートする。これによって、次の浮動小数点命令、例えば浮動小数点加算命令１５１４をＥ１ステージ２２６に伝達するのが許容される。

【００７０】矢印１５１８は、浮動小数点除算命令１５１０のＥ１ステージの実行に要する１２サイクルの後に続くＦＰＵパイプライン２１０のＥ１ステージに浮動小数点加算命令が進むことを示す。同様に矢印１５２２は、浮動小数点除算命令１５１０のＥ１ステージ２２６における第１２番目のサイクルの後に続くＣＰＵパイプライン２１４のＤＩデコードステージ２３８に第５番目の命令すなわちＣＰＵ加算命令が進むことを示す。このように、通常よりも実行サイクル数の長い“ＦＤＩＶ”のような浮動小数点命令が実行される場合にも、前記ビジー信号によって適切にストールされることにより、ＣＰＵパイプラインとＦＰＵパイプラインは同期を保つことができる。仮に、浮動小数点加算命令１５１４の実行によって例外が発生される場合には、浮動小数点加算命令１５１４は１３サイクル目１５２６で例外を発生させることになる。したがって、その例外はプログラムの実行順序を乱さない。即ち、その例外は正確である。なぜならば、浮動小数点加算命令１５１４の次の命令は実行の機会をまだ持っていないからである。すなわち、１３サイクル目で浮動小数点加算命令１５１４が例外を生じたとき、後続の命令は、まだ実行ステージに進んでいないからである。これも、ＣＰＵパイプラインとＦＰＵパイプラインが同期化を保っているからである。

【００７１】図１７はビジー信号経路１２４にビジー信号１５０８をアサートするためのビジー信号回路を示す。特に、最初の命令がＦＰＵパイプライン２１０のデコードステージ（Ｄｆ）２２２に入力されたという事実を示す信号は論理的なアンドゲート（ＡＮＤ）１６１８に適用される。更には、デコードステージ（Ｄｆ）２２２は、最初の浮動小数点命令、この例の場合浮動小数点除算命令がＥ２ステージに入力されたという事実を示す信号を適用する。しかしながら、その信号は、アンドゲート１６１８によって処理される前に、インバータ１６２６によって反転される。そして、最初の浮動小数点命令がＥ１ステージ２２６に入力されたという事実を示す信号は、論路的なオアゲート（ＯＲ）１６３８の入力１６３４に適用される。同様に、アンドゲート１６１８の出力も同様にオアゲート１６５０の他方の入力１６４６に適用される。オアゲート１６３８の出力１６５０はＣＰＵパイプライン２１４のデコードステージ（Ｄｉ）２３８にビジー信号１５０８を供給し、それによって、Ｃ

ＰＵパイプライン２１４にストールを発生させることができると共に、ＦＰＵパイプライン２１０のＤｆステージで命令を繰り返すことができる。

【００７２】以上本発明者によってなされた発明を実施形態に基づいて具体的に説明したが、本発明はそれに限定されるものではなく、その要旨を逸脱しない範囲において種々変更可能であることは言うまでもない。例えば、ＣＰＵとＦＰＵのパイプライン段数は夫々５段に限定されず適宜変更可能である。また、本発明はＲＩＳＣプロセッサに限定されずＣＩＳＣプロセッサなど、その他のアーキテクチャを有するマイクロプロセッサに適用可能である。また、この明細書においてマイクロプロセッサは、マイクロコンピュータ、シングルチップマイクロコンピュータ、データプロセッサを含む概念として用いている。

【００７３】

【発明の効果】本願において開示される発明のうち代表的なものによって得られる効果を簡単に説明すれば下記の通りである。

【００７４】すなわち、本発明に係るマイクロプロセッサは、命令実行に際して、ＦＰＵパイプラインとＣＰＵパイプラインを並列的に進行させ、浮動小数点パイプラインとＣＰＵパイプラインとのパイプライン段数同一とすることによって、ＣＰＵパイプラインとＦＰＵパイプラインのフロー制御などのパイプライン制御のためのリソースを共有させることができる。このリソースの共有化は、パイプライン制御のための回路規模を縮小可能にする。これらは、浮動小数点パイプラインとプロセッサパイプラインとを同期化し易くする。

【００７５】パイプラインに対してフリーズやストールを必要とする場合、双方のパイプラインにストールとフリーズの効果をもたらすようにパイプラインの一方がストール又はフリーズを制御することによってＦＰＵパイプラインとＣＰＵパイプラインの同期化を達成することができる。

【００７６】ＦＰＵＬレジスタのようなレジスタをＣＰＵとＦＰＵが共有することによってＣＰＵとＦＰＵとの間のデータ交換を簡単に且つ高速に行なうことができる。

【００７７】浮動小数点比較命令の後に続く条件分岐に対するスループットを向上させることができる。

【００７８】浮動小数点除算命令においてもＦＰＵパイプラインとＣＰＵパイプラインとを同期化できる。

【図面の簡単な説明】

【図１】本発明によるマイクロプロセッサの一例を示すブロック図である。

【図２】ＦＰＵとＣＰＵのパイプラインの関係を示す説明図である。

【図３】命令“ＬＤＳ　Ｒｍ，ＦＰＵＬ”の実行に際してのＦＰＵパイプライン及びＣＰＵのパイプライン、そして双方のパイプライン間でのデータ交換のタイミングを示す説明図である。

【図４】命令“ＳＴＳ　ＦＰＵＬ，Ｒｎ”の実行に際してのＦＰＵパイプライン及びＣＰＵのパイプライン、そして双方のパイプライン間でのデータ交換のタイミングを示す説明図である。

【図５】ＦＰＵのパイプラインを更に詳細に示す説明図である。

【図６】パイプラインリソースの共有によるＦＰＵパイプラインとＣＰＵパイプラインの同期の一例を示す説明図である。

【図７】ストールの第１の形態であるロード・ユース・ストールを例示するパイプラインダイヤグラムである。

【図８】ストールのもう一つの形態であるメモリアクセス・コンフリクト・ストールを例示するパイプラインダイヤグラムである。

【図９】ストール信号を生成する回路を例示するブロック図である。

【図１０】ＦＰＵパイプラインとＣＰＵパイプラインの双方をストールすることによりＣＰＵデコードステージがどうやって同期を維持するかを例示する説明図である。

【図１１】ＦＰＵパイプラインのデコードステージを例示する説明図である。

【図１２】ＦＰＵパイプラインの第１実行ステージＥ１を例示する説明図である。

【図１３】ＦＰＵの詳細回路を例示するブロック図である。

【図１４】ＣＰＵ分岐命令が後に続く浮動小数点比較命令の手順のためのＦＰＵとＣＰＵパイプラインの同期化を例示する説明図である。

【図１５】Ｅ１ステージからＥステージにＴビットをバイパスするための回路の一例を示すブロック図である。

【図１６】例外を正確に保つためのＦＰＵとＣＰＵのパイプラインの同期化の一例を示す説明図である。

【図１７】ビジー信号経路にビジー信号をアサートするためのビジー信号回路の一例を示すブロック図である。

【図１８】レジスタコンフリクトの説明図である。

【図１９】ＣＰＵパイプラインとＦＰＵパイプラインのリソース共有状態の一例を概念的に示す説明図である。

【図２０】図１のマイクロプロセッサがサポートする命令セットの内、浮動小数点命令と浮動小数点命令に関連するＣＰＵ命令とを列挙した説明図である。

【図２１】浮動小数点命令“ＦＡＢＳ”の説明図である。

【図２２】浮動小数点命令“ＦＡＤＤ”の説明図である。

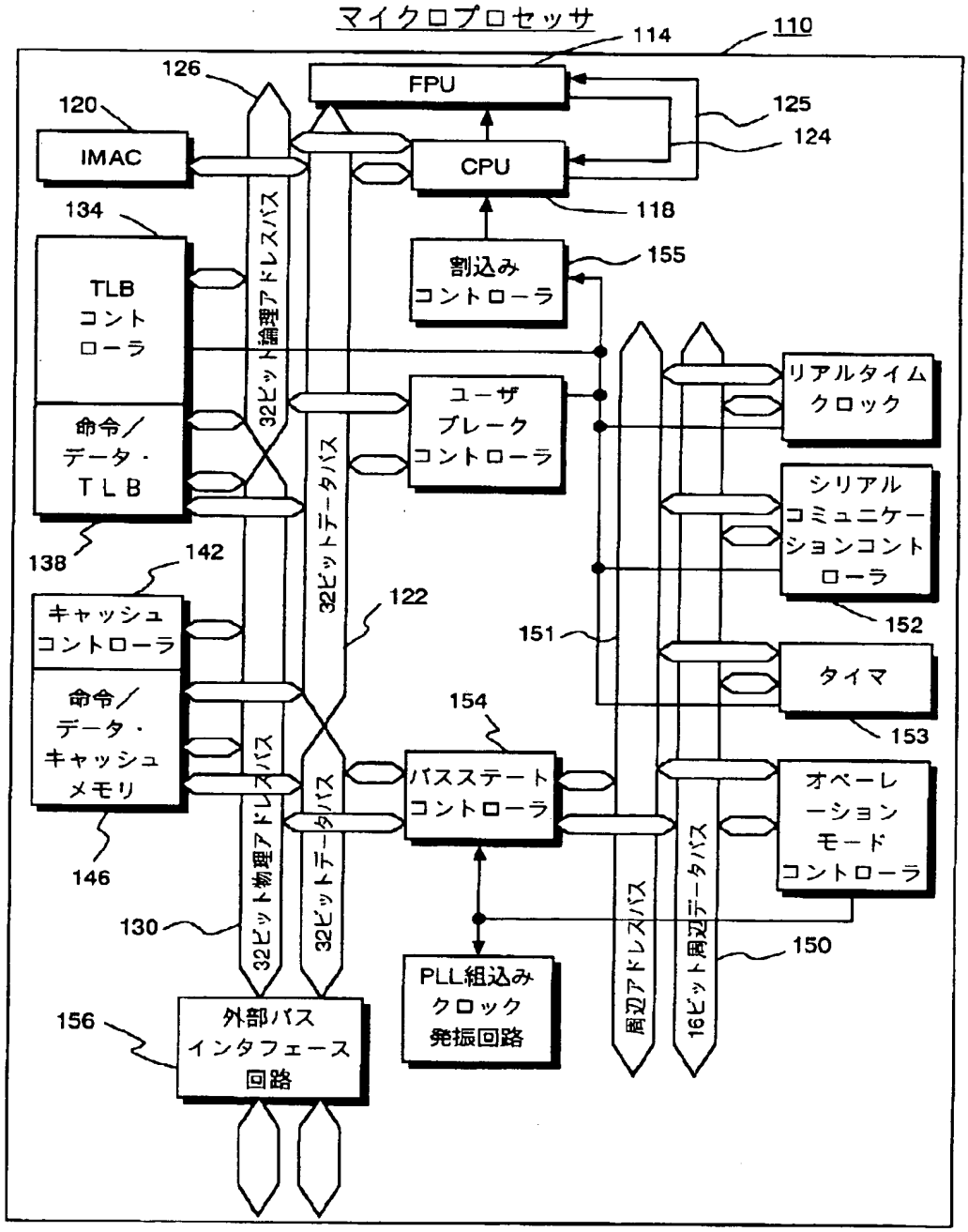
【図２３】図２２に続く浮動小数点命令“ＦＡＤＤ”の説明図である。

【図２４】図２３に続く浮動小数点命令“ＦＡＤＤ”の

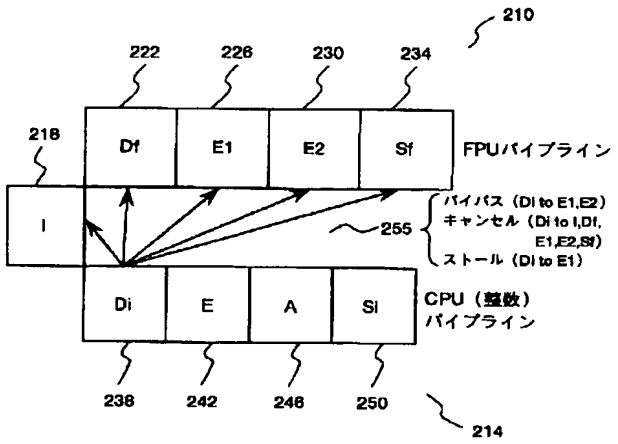
説明図である。
【図２５】浮動小数点命令“ＦＣＭＰ”の説明図である。
【図２６】図２５に続く浮動小数点命令“ＦＣＭＰ”の説明図である。
【図２７】図２６に続く浮動小数点命令“ＦＣＭＰ”の説明図である。
【図２８】浮動小数点命令“ＦＤＩＶ”の説明図である。
【図２９】図２８に続く浮動小数点命令“ＦＤＩＶ”の説明図である。
【図３０】浮動小数点命令“ＦＬＤＩ０”の説明図である。
【図３１】浮動小数点命令“ＦＬＤＩ１”の説明図である。
【図３２】浮動小数点命令“ＦＬＤＳ”の説明図である。
【図３３】浮動小数点命令“ＦＭＵＬ”の説明図である。
【図３４】図３３に続く浮動小数点命令“ＦＭＵＬ”の説明図である。
【図３５】浮動小数点命令“ＦＮＥＧ”の説明図である。
【図３６】浮動小数点命令“ＦＳＱＲＴ”の説明図である。
【図３７】浮動小数点命令“ＦＳＴＳ”の説明図である。
【図３８】浮動小数点命令“ＦＳＵＢ”の説明図である。
【図３９】図３８に続く浮動小数点命令“ＦＳＵＢ”の説明図である。
【図４０】図３９に続く浮動小数点命令“ＦＳＵＢ”の説明図である。
【図４１】浮動小数点命令“ＦＴＲＣ”の説明図である。
【図４２】図４１に続く浮動小数点命令“ＦＴＲＣ”の説明図である。
【図４３】浮動小数点命令“ＦＴＳＴ”の説明図である。
【図４４】ＣＰＵ命令“ＬＤＳ”の説明図である。
【図４５】図４４に続くＣＰＵ命令“ＬＤＳ”の説明図である。
【図４６】ＣＰＵ命令“ＳＴＳ”の説明図である。
【図４７】図４６に続くＣＰＵ命令“ＳＴＳ”の説明図である。
【図４８】浮動小数点命令“ＦＬＯＡＴ”の説明図である。
【図４９】浮動小数点命令“ＦＭＡＣ”の説明図である。
【図５０】図４９に続く浮動小数点命令“ＦＭＡＣ”の説明図である。
【図５１】浮動小数点命令“ＦＭＯＶの説明図である。
【図５２】図５１に続く浮動小数点命令“ＦＭＯＶの説明図である。
【符号の説明】
１１０　マイクロプロセッサ
１１４　ＦＰＵ
１１８　ＣＰＵ
２１０　ＦＰＵパイプライン
２１４　ＣＰＵパイプライン
２１８　命令フェッチステージ（Ｉ）
２２２　ＦＰＵパイプラインのデコードステージ（Ｄｆ）
２２６，２３０　ＦＰＵパイプラインの実行ステージ（Ｅ１，Ｅ２）
２３４　ＦＰＵパイプラインのライトバックステージ（Ｓｆ）
２３８　ＣＰＵパイプラインのデコードステージ（Ｄｉ）
２４２　ＣＰＵパイプラインの実行ステージ（Ｅ）
２４６　ＣＰＵパイプラインのメモリアクセスステージ（Ａ）
２５０　ＣＰＵパイプラインのライトバックステージ（Ｓｉ）
２５５　バイパス、ストール、キャンセル経路
ＦＰＵＬ　コミュニケーションレジスタ
４２８　ＣＰＵデコードステージからのバイパス信号
５１８　ＣＰＵデコーダからのバイパス信号
６２２　ストール信号
７２５　ストール信号
９１４　ストール信号
１４１０　Ｅ１ステージからのＴビット
１５０８　ビジー信号
１６０８　ビジー信号

【図１】

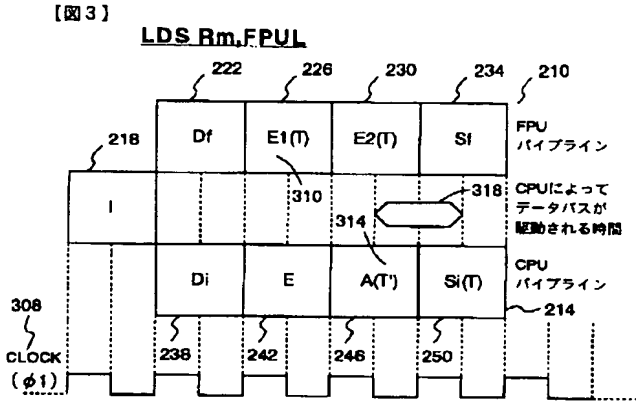
【図１】


マイクロプロセッサ
110
114
FPU
126
120
IMAC
125
124
CPU
118
134
TLB
コント
ローラ
割込み
コントローラ
155
32ビット論理アドレスバス
32ビットデータバス
命令／
データ・
TLB
ユーザ
ブレーク
コントローラ
リアルタイム
クロック
シリアル
コミュニケー
ションコント
ローラ
152
138
142
キャッシュ
コントローラ
122
151
タイマ
153
命令／
データ・
キャッシュ
メモリ
154
バスステート
コントローラ
オペレ
ーション
モード
コントローラ
146
130
32ビット物理アドレスバス
32ビットデータバス
周辺アドレスバス
16ビット周辺データバス
150
PLL組込み
クロック
発振回路
156
外部バス
インタフェース
回路

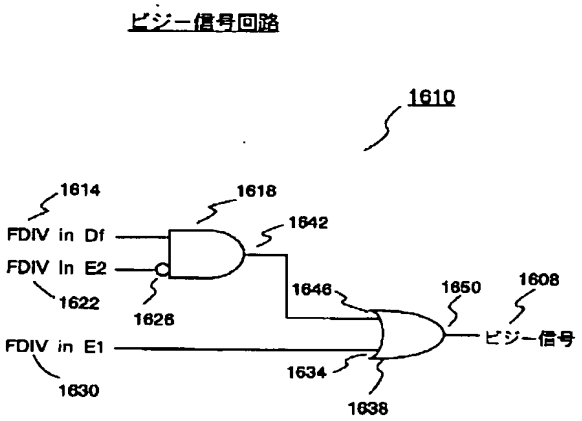
【図２】



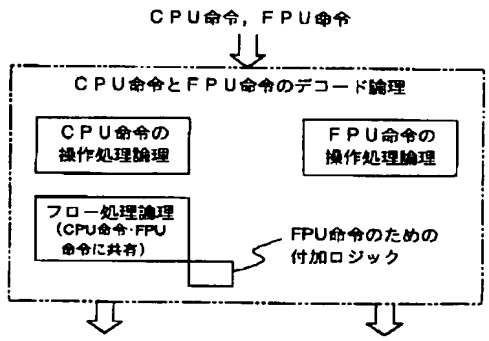
【図３】



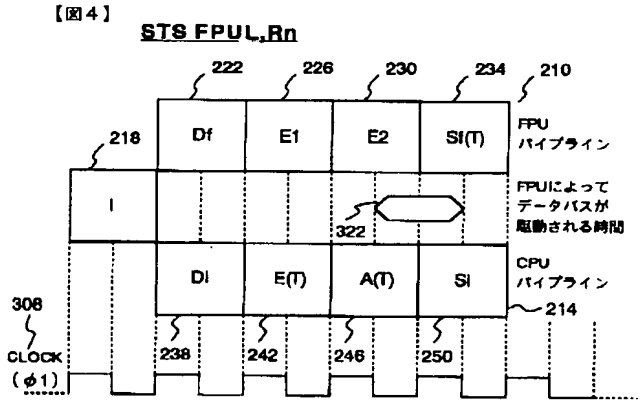
【図１７】



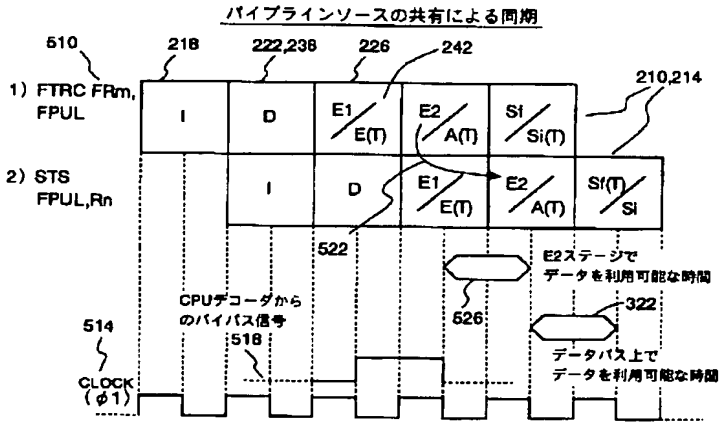
【図１９】



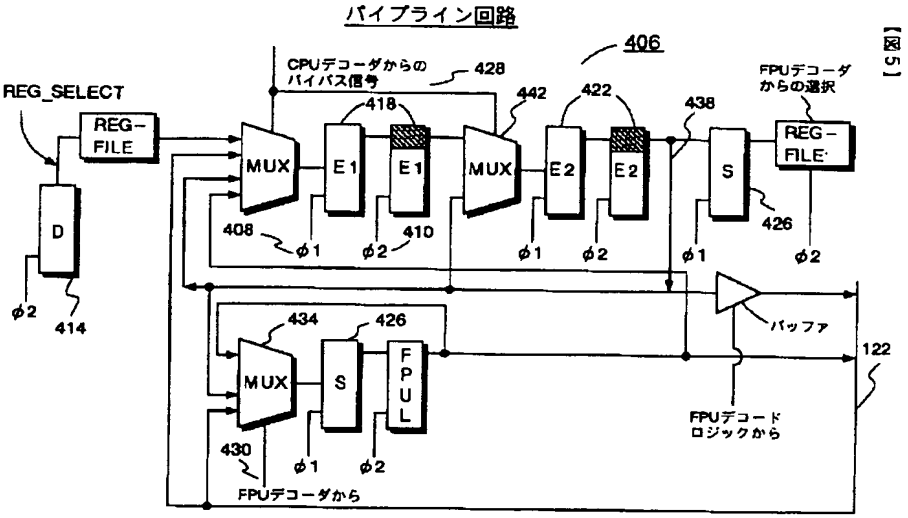
【図４】



【図６】

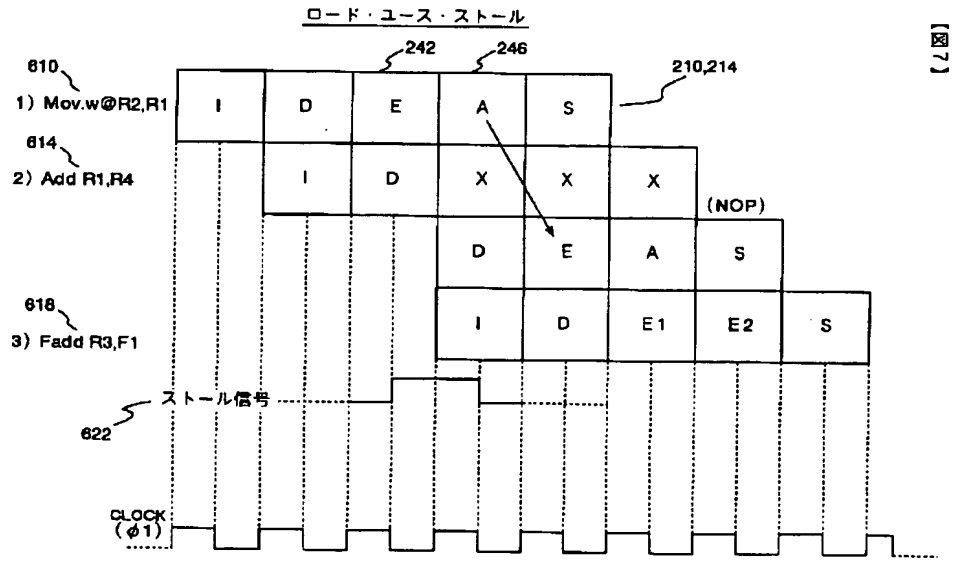


【図５】


パイプライン回路
406
CPUデコーダからの
バイパス信号
428
FPUデコーダ
からの選択
REG_SELECT
REG-
FILE
MUX
E1
E1
MUX
E2
E2
S
REG-
FILE
418
442
422
438
426
D
408
φ1
φ2
410
φ1
φ2
φ1
φ2
φ2
414
バッファ
122
434
426
MUX
S
FPUL
FPUデコード
ロジックから
430
φ1
φ2
FPUデコーダから

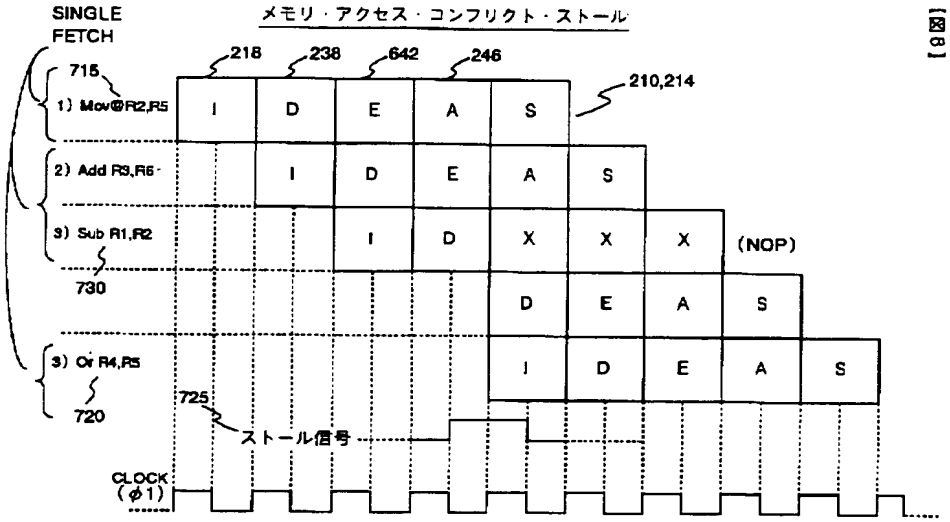
【図５】

【図７】

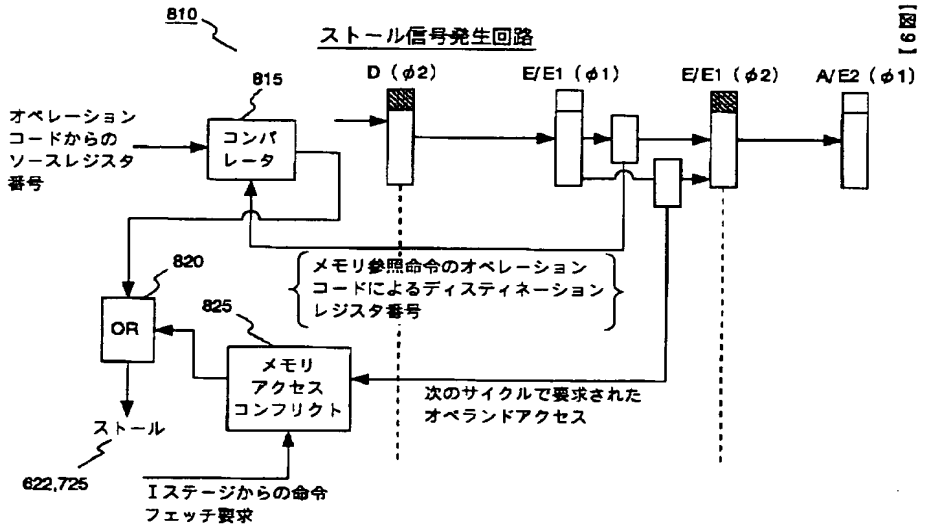

ロード・ユース・ストール
242
246
210,214
610
1) Mov.w@R2,R1
I
D
E
A
S
614
2) Add R1,R4
I
D
X
X
X
(NOP)
D
E
A
S
618
3) Fadd R3,F1
I
D
E1
E2
S
ストール信号
622
CLOCK
(φ1)

【図７】

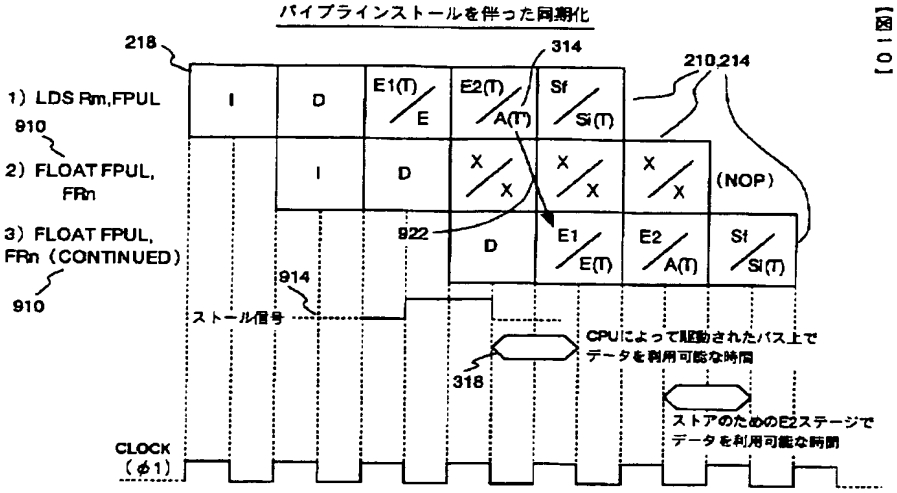
【図８】


SINGLE FETCH
メモリ・アクセス・コンフリクト・ストール
218
238
642
246
210,214
715
1) Mov@R2,R5
2) Add R3,R6
3) Sub R1,R2
730
3) Or R4,R5
720
725
ストール信号
CLOCK (φ1)
I
D
E
A
S
X
(NOP)

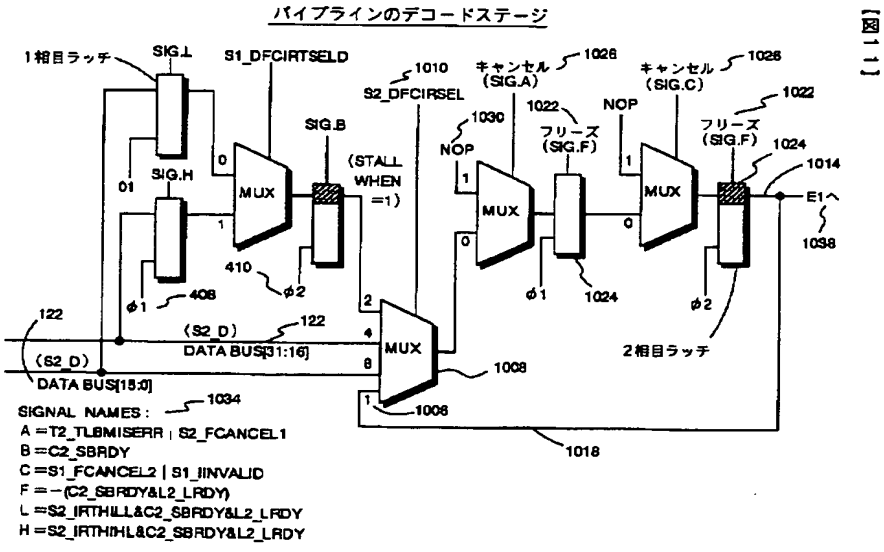
【図９】


810
ストール信号発生回路
815
D (φ2)
E/E1 (φ1)
E/E1 (φ2)
A/E2 (φ1)
オペレーション コードからの ソースレジスタ 番号
コンパ レータ
820
OR
825
メモリ アクセス コンフリクト
メモリ参照命令のオペレーション コードによるディスティネーション レジスタ番号
次のサイクルで要求された オペランドアクセス
ストール
622,725
Iステージからの命令 フェッチ要求

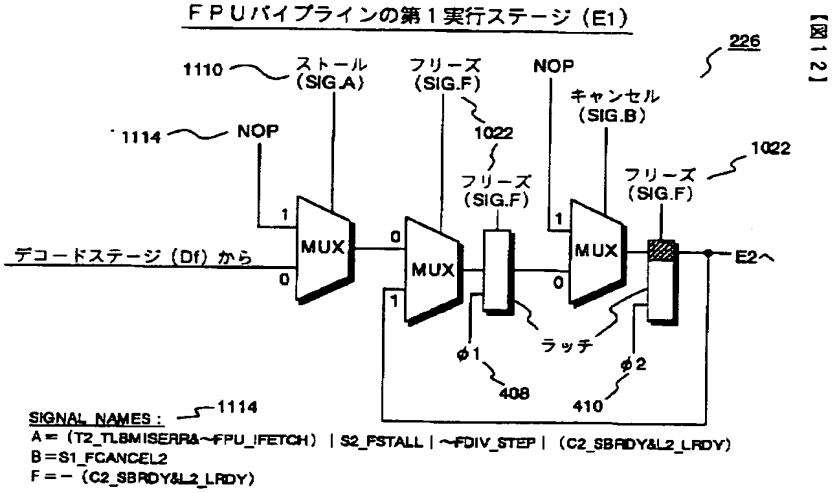
【図１０】


パイプラインストールを伴った同期化
218
314
210,214
1) LDS Rm,FPUL
910
2) FLOAT FPUL, FRn
3) FLOAT FPUL, FRn (CONTINUED)
910
I
D
E1(f) E
E2(f) A(f)
Sf Si(f)
X X
(NOP)
922
E1 E(f)
E2 A(f)
914
ストール信号
CPUによって駆動されたバス上でデータを利用可能な時間
318
ストアのためのE2ステージでデータを利用可能な時間
CLOCK (φ1)

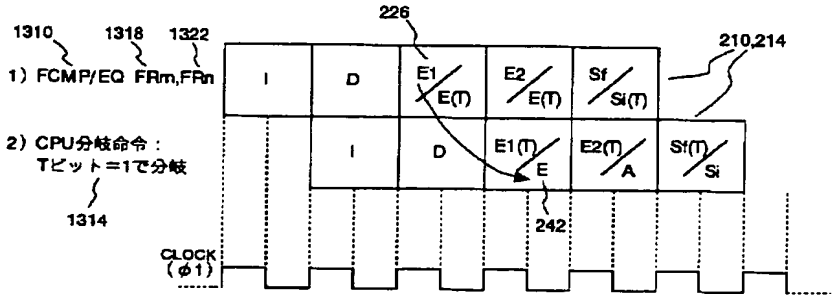
【図１１】


パイプラインのデコードステージ
1相目ラッチ
SIG.L
S1_DFCIRTSELD
1010
S2_DFCIRSEL
キャンセル (SIG.A)
1026
キャンセル (SIG.C)
1026
1022
フリーズ (SIG.F)
NOP
1030
SIG.B
(STALL WHEN =1)
SIG.H
MUX
410
φ2
φ1
408
122
(S2_D)
DATA BUS[31:16]
DATA BUS[15:0]
1008
1006
1018
1024
2相目ラッチ
1014
E1へ
1038
SIGNAL NAMES :
1034
A =T2_TLBMISERR | S2_FCANCEL1
B =C2_SBRDY
C =S1_FCANCEL2 | S1_IINVALID
F =−(C2_SBRDY&L2_LRDY)
L =S2_IRTHILL&C2_SBRDY&L2_LRDY
H =S2_IRTHIHL&C2_SBRDY&L2_LRDY

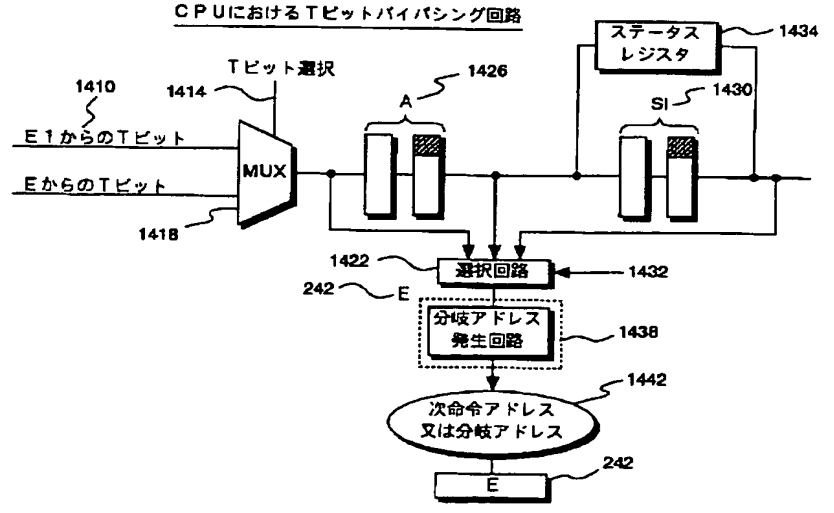
【図１２】


ＦＰＵパイプラインの第１実行ステージ（E1）
1110
ストール
(SIG.A)
フリーズ
(SIG.F)
NOP
226
キャンセル
(SIG.B)
1114
NOP
1022
フリーズ
(SIG.F)
1022
フリーズ
(SIG.F)
MUX
MUX
MUX
デコードステージ（Df）から
E2へ
ラッチ
φ1
φ2
408
410
SIGNAL NAMES :
1114
A＝（T2_TLBMISERR&~FPU_IFETCH）｜S2_FSTALL｜~FDIV_STEP｜（C2_SBRDY&L2_LRDY）
B＝S1_FCANCEL2
F＝－（C2_SBRDY&L2_LRDY）

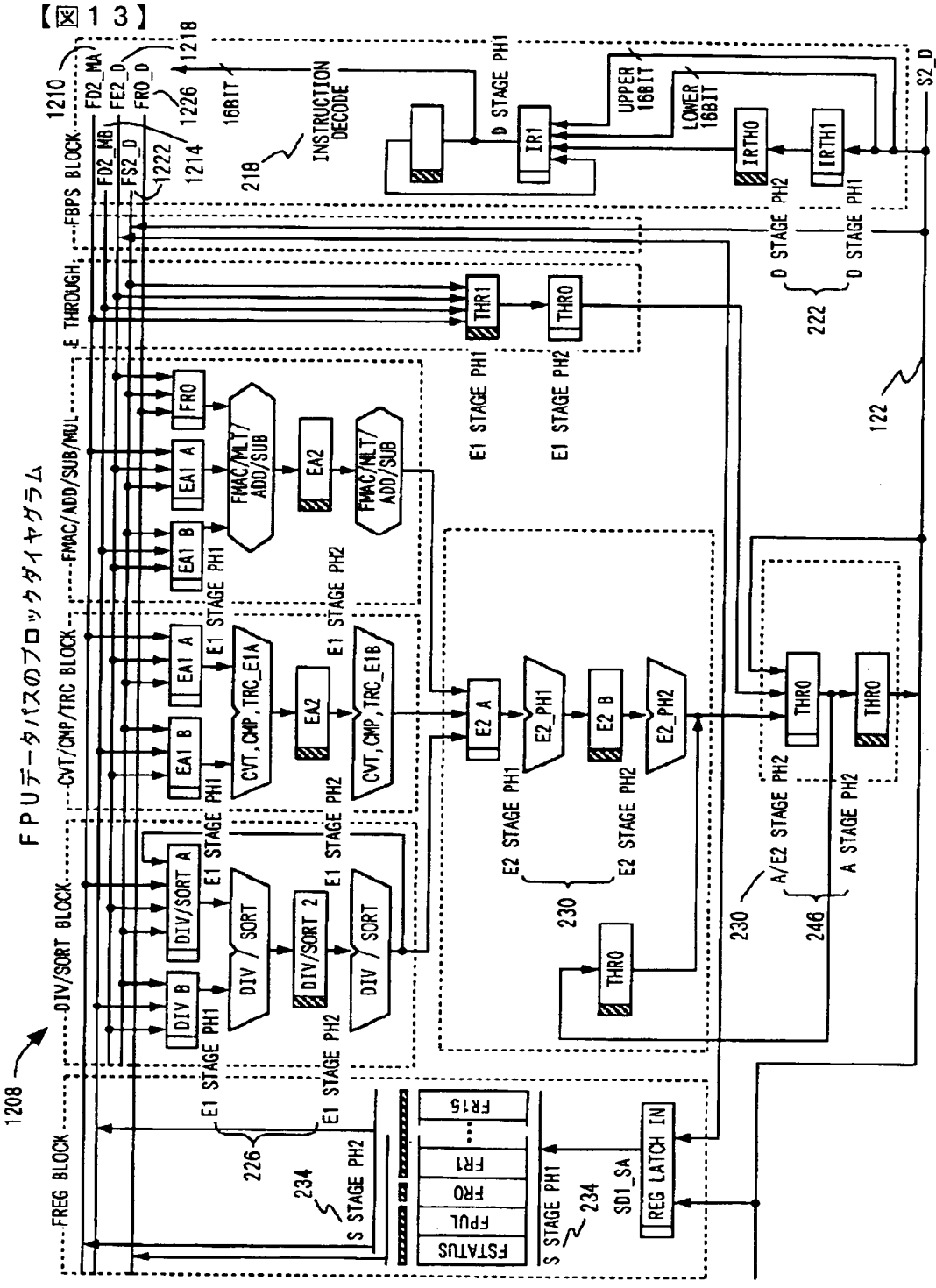
【図１４】


Ｔビットバイパシング
1310
1318
1322
226
210,214
1) FCMP/EQ FRm,FRn
I
D
E1
E(T)
E2
E(T)
Sf
Si(T)
2) CPU分岐命令：
Tビット＝1で分岐
1314
I
D
E1(T)
E
E2(T)
A
Sf(T)
Si
242
CLOCK
(φ1)

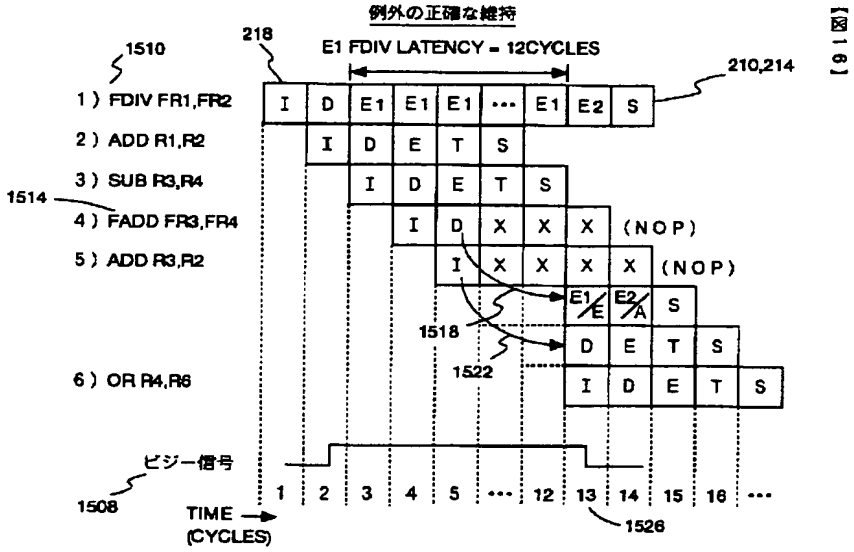
【図１５】


ＣＰＵにおけるＴビットバイパシング回路
ステータス
レジスタ
1434
Tビット選択
1414
1426
A
SI
1430
1410
E１からのTビット
Eからのビット
MUX
1418
1422
選択回路
1432
242
E
分岐アドレス
発生回路
1438
1442
次命令アドレス
又は分岐アドレス
E
242

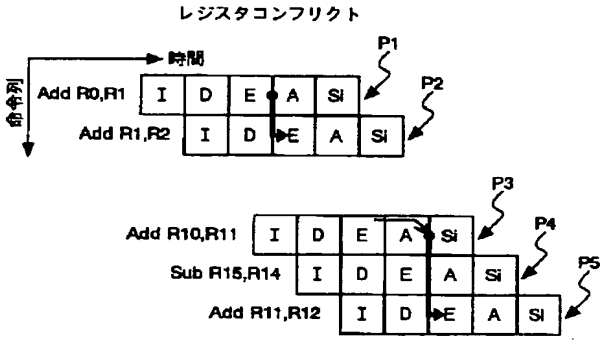
【図１３】


FPUデータパスのブロックダイヤグラム
1208
1210
1218
1226
1214
1222
218
16BIT
INSTRUCTION DECODE
FD2_MA
FD2_MB
FE2_D
FS2_D
FR0_D
FBPS BLOCK
E THROUGH
FMAC/ADD/SUB/MUL
CVT/CMP/TRC BLOCK
DIV/SORT BLOCK
FREG BLOCK
D STAGE PH1
D STAGE PH2
IR1
IRTH0
IRTH1
UPPER 16BIT
LOWER 16BIT
S2_D
222
122
THR1
THR0
E1 STAGE PH1
E1 STAGE PH2
EA1 A
EA1 B
EA2
FR0
FMAC/MLT/ADD/SUB
CVT,CMP,TRC_E1A
CVT,CMP,TRC_E1B
DIV/SORT A
DIV/SORT 2
DIV B
DIV
SORT
226
234
S STAGE PH2
S STAGE PH1
FR15
FR1
FR0
FPUL
FSTATUS
SD1_SA
REG LATCH IN
E2 A
E2_PH1
E2 B
E2_PH2
E2 STAGE PH1
E2 STAGE PH2
230
A/E2 STAGE PH2
A STAGE PH2
246

【図１６】



【図１８】



【図２０】

浮動小数点命令

Operation	op code	mnemonic
Floating Move (Load)	FNM8	FMOV.S @Rm,FBn
Floating Move (Store)	FNMA	FMOV.S FBm,@Rn
Floating Move (Restore)	FNM9	FMOV.S @Rm+,FBn
Floating Move (Save)	FNMB	FMOV.S FBm,@-Rn
Floating Move (Load with index)	FNM6	FMOV.S @(R0,Rm),FBn
Floating Move (Store with index)	FNM7	FMOV.S FBm,@(R0,Rn)
Floating Move (in register file)	FNMC	FMOV FBm,FBn
Floating Load Immediate 0	FN8D	FLDI0 FBn
Floating Load Immediate 1	FN9D	FLDI1 FBn
Floating Add	FNM0	FADD FBm,FBn
Floating Subtract	FNM1	FSUB FBm,FBn
Floating Multiply	FNM2	FMUL FBm,FBn
Floating Divide	FNM3	FDIV FBm,FBn
Floating Multiply Accumulate	FNME	FMAC FB0,FBm,FBn
Floating Compare Equal	FNM4	FCMP/EQ FBm,FBn
Floating Compare Greater Than	FNM5	FCMP/GT FBm,FBn
Floating Test NaN	FN7D	FTST/ NAN FBn
Floating Negate	FN4D	FNEG FBn
Floating Absolute Value	FN5D	FABS FBn
Floating Square Root	FN6D	FSQRT FBn
Floating Convert from Integer	FN2D	FLOAT FPUL,FBn
Floating Truncate and Convert to Integer	FN3D	FTRC FBm,FPUL
Floating Store from System Register FPUL	FN0D	FSTS FPUL,FBn
Floating Load to System Register FPUL	FN1D	FLDS FBm,FPUL

FPU関連CPU命令

Operation	op code	mnemonic
Load from System Register FPUL	4N5A	LDS Rm,FPUL
Restore System Register FPUL	4N56	LDS.L @Rm+,FPUL
Load from System Register FPSCR	4N6A	LDS Rm,FPSCR
Restore System Register FPSCR	4N66	LDS.L @Rm+,FPSCR
Store to System Register FPUL	0N5A	STS FPUL,Rn
Save System Register FPUL	4N52	STS.L FPUL,@-Rn
Store to System Register FPSCR	0N6A	STS FPSCR,Rn
Save System Register FPSCR	4N62	L FPSCR,@-Rn

FABS (Floating Point Absolute Value): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FABS FRn	|FRn| -> FRn	1111nnnn01011101	2	1	-

Description: Takes floating-point-arithmetic absolute value of the content of floating point register FRn. And the result of this operation is written on the FRn.

Operation:

```
FABS(float *FRn)  /* FABS FRn */
{
  clear_cause_VZ();
  case(data_type_of(FRn))      {
      NORM :  if(sign_of(FRn) == 0) *FRn =  *FRn;
              else                  *FRn = -*FRn;
                                 break;
      PZERO:
      NZERO:       zero(FRn,0);       break;
      PINF :
      NINF :        inf(FRn,0);       break;
      qNaN :       qnan(FRn);  break;
      sNaN :    invalid(FRn);  break;
  }
  pc += 2;
}
```

FABS Special Cases

FRn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
FABS (FRn)	ABS	+0	+0	+INF	+INF	qNaN	Invaild

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

【図２１】

【図２１】

特開平９－２１２３５９

FADD (Floating Point Add): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FADD FRm, FRn	FRn+FRm -> FRn	1111nnnnmmmm0000	2	1	-

Description: Floating-point-arithmetically adds the contents of floating point registers FRm and FRn. And the result of this operation is written on the FRn.

Operation:

```
FADD(float *FRm,*FRn)    /* FADD FRm,FRn */
{
         clear_cause_VZ();
         if((data_type_of(FRm) == sNaN) ||
            (data_type_of(FRn) == sNaN))  invalid(FRn);
    else if((data_type_of(FRm) == qNaN) ||
            (data_type_of(FRn) == qNaN))  qnan(FRn);
   else case(data_type_of(FRm))      {
      NORM :
        case(data_type_of(FRn))      {
            PINF :       inf(FRn,0);       break;
            NINF :       inf(FRn,1);       break;
          default:       *FRn = *FRn + *FRm;     break;
        }                            break;
      PZERO:
        case(data_type_of(FRn))      {
            NORM :       *FRn = *FRn + *FRm;     break;
            PZERO:
            NZERO:      zero(FRn,0);       break;
            PINF :       inf(FRn,0);       break;
            NINF :       inf(FRn,1);       break;
        }                            break;
```

【図22】

【図22】

【図２３】

【図２３】

```
        NZERO:
          case(data_type_of(FRn))        {
              NORM :        *FRn = *FRn + *FRm;      break;
              PZERO:       zero(FRn,0);        break;
              NZERO:       zero(FRn,1);        break;
              PINF :        inf(FRn,0);        break;
              NINF :        inf(FRn,1);        break;
          }                                  break;
        PINF :
          case(data_type_of(FRn))        {
              NINF :  invalid(FRn);          break;
            default:        inf(FRn,0);      break;
          }                                  break;
        NINF :
          case(data_type_of(FRn))        {
              PINF :  invalid(FRn);          break;
            default:        inf(FRn,1);      break;
          }                                  break;
    }
    pc += 2;
}
```

【図２４】

【図２４】

FADD Special Cases

FRm \ FRn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	ADD				-INF		
+0		+0					
-0			-0				
+INF				+INF	Invalid		
-INF	-INF			Invalid	-INF		
qNaN						qNaN	
sNaN							Invalid

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

【図25】

FCMP (Floating Point Compare): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FCMP/EQ FRm, FRn	(FRn==FRm)? 1:0 ->T	1111nnnnmmmm0100	2	1	1/0
FCMP/GT FRm, FRn	(FRn> FRm)? 1:0 ->T	1111nnnnmmmm0101	2	1	1/0

Description: Floating-point-arithmetically compares between the contents of floating point registers FRm and FRn. And the result of this operation, true/false, is written on the T bit.

Operation:

```
FCMP_EQ(float *FRm,*_FRn)       /* FCMP/EQ FRm,_FRn */
{
      clear_cause_VZ();
      if(fcmp_chk(FRm,FRn) == INVALID) {  fcmp_invalid(0); }
      else if(fcmp_chk(FRm,FRn) == EQ)    T = 1;
      else                                T = 0;
      pc += 2;
}
FCMP_GT(float *FRm,*FRn)        /* FCMP/GT FRm,FRn */
{
      clear_cause_VZ();
      if(fcmp_chk(FRm,FRn) == INVALID) {  fcmp_invalid(0); }
      else if(fcmp_chk(FRm,FRn) == GT)    T = 1;
      else                                T = 0;
      pc += 2;
}
fcmp_chk(float *FRm,*FRn)
```

【図25】

特開平9－212359

【図２６】

【図２６】

```
{
        if((data_type_of(FRm) == sNaN) ||
           (data_type_of(FRn) == sNaN))  return(INVALID);
    else if((data_type_of(FRm) == qNaN) ||
           (data_type_of(FRn) == qNaN))  return(UO);
   else case(data_type_of(FRm))      {
      NORM :  case(data_type_of(FRn))      {
                  PINF : return(GT);              break;
                  NINF : return(NOTGT);           break;
                default:                          break;
              }                                  break;
      PZERO:
      NZERO:  case(data_type_of(FRn))      {
                  PZERO:
                  NZERO: return(EQ);              break;
                  PINF : return(GT);              break;
                  NINF : return(NOTGT);           break;
                default:                          break;
              }                                  break;
      PINF :  case(data_type_of(FRn))      {
                  PINF : return(EQ);              break;
                default: return(NOTGT);           break;
              }                                  break;
      NINF :  case(data_type_of(FRn))      {
                  NINF : return(EQ);              break;
                default: return(GT);              break;
              }                                  break;
  }
  if(*FRn==*FRm)    return(EQ);
  else if(*FRn>*FRm) return(GT);
  else               return(NOTGT);
}
fcmp_invalid(int cmp_flag)
```

```
  set_V();
  if((FPSCR & ENABLE_V) == 0) T = cmp_flag;
}
```

FCMP Special Cases

FRm \ FRn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	CMP			GT	! GT		
+0		EQ					
-0							
+INF	! GT			EQ			
-INF	GT				EQ		
qNaN	UO						
sNaN	Invalid						

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

Note: IEEE defines the independly 4 conditions of caparison. But FPU support FCMP/EQ and FCMP/GT only. But FPU can supprot all conditions using the conbination of BT/BF , FTST/NAN and these 2 FCMPs.

Unorder FRm, FRn	ftst/nan FRm ; bf ; ftst/nan FRn ; bf
(FRm == FRn)	fcmp/eq FRm, FRn ; bt
(FRm != FRn)	fcmp/eq FRm, FRn ; bf
(FRm < FRn)	fcmp/gt FRm, FRn ; bt
(FRm <= FRn)	fcmp/gt FRn, FRm ; bf
(FRm >FRn)	fcmp/gt FRn, FRm ; bt
(FRm >= FRn)	fcmp/gt FRm, FRn ; bf

【図２７】

【図２７】

特開平９－２１２３５９

FDIV (Floating Point Divide): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FDIV FRm,FRn	FRn/FRm -> FRn	1111nnnnmmmm0011	13	12	-

Description: Floating-point-arithmetically divides the content of floating point register FRn by the content of floating point register FRm. And the result of this operation is written on the FRn.

Operation:

```
FDIV(float *FRm,*FRn)  /* FDIV FRm,FRn */
{
        clear_cause_VZ();
        if((data_type_of(FRm) == sNaN) ||
           (data_type_of(FRn) == sNaN))  invalid(FRn);
   else if((data_type_of(FRm) == qNaN) ||
           (data_type_of(FRn) == qNaN))  qnan(FRn);
  else case(data_type_of(FRm))      {
    NORM :
      case(data_type_of(FRn))       {
          PINF :
          NINF :  inf(FRn,sign_of(FRm)^sign_of(FRn));     break;
        default:  *FRn = *FRn / *FRm;                     break;
      }                                   break;
    PZERO:
    NZERO:
      case(data_type_of(FRn))       {
          PZERO:
          NZERO:  invalid(FRn);                           break;
        default:  dz(FRn,sign_of(FRm)^sign_of(FRn));      break;
      }                                   break;
```

【図28】

【図28】

特開平9－212359

【図２９】

```
        PINF :
        NINF :
          case(data_type_of(FRn))      {
              PINF :
              NINF :  invalid(FRn);                                  break;
            default:  zero(FRn,sign_of(FRm)^sign_of(FRn));           break;
          }                                        break;
    }
    pc += 2;
}
```

【図２９】

FDIV Special Cases

FRm \ FRn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	DIV	0		INF			
+0	DZ	Invalid		DZ			
-0							
+INF	0	+0	-0	Invalid			
-INF		-0	+0				
qNaN	qNaN						
sNaN	Invalid						

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

【図３４】

```
        NINF :
          case(data_type_of(FRn))      {
              PZERO:
              NZERO:  invalid(FRn);
            default:  inf(FRn,sign_of(FRm)^sign_of(FRn));            break;
          }                                        break;
    }
    pc += 2;
}
```

【図３４】

FMUL Special Cases

FRm \ FRn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	MUL	0		INF			
+0	0	+0	-0	Invalid			
-0		-0	+0				
+INF	INF	Invalid		+INF	-INF		
-INF				-INF	+INF		
qNaN	qNaN						
sNaN	Invalid						

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

FLDI0 (Floating Point Load Immediate 0): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FLDI0 FRn	0x00000000 -> FRn	1111nnnn10001101	2	1	-

Description: Loads floating point zero (0x00000000) to the floating point register FRn.

Operation:

```
FLDI0(float *FRn) /* FLDI0 FRn */
{
      *FRn  = 0x00000000;
       pc += 2;
}
```

Exceptions:
None

【図30】

特開平9－212359

FLDI1 (Floating Point Load Immediate 1): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FLDI1 FRn	0x3F800000 -> FRn	1111nnnn10011101	2	1	-

Description: Loads floating point one (0x3F800000) to the floating point register FRn.

Operation:

```
FLDI1(float *FRn) /* FLDI1 FRn */
{
      *FRn  = 0x3F800000;
       pc += 2;
}
```

Exceptions:
None

【図３１】

特開平９－２１２３５９

【図32】

FLDS (Floating Point Load to System Register): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FLDS FRm, FPUL	FRm -> FPUL	1111nnnn00011101	2	1	-

Description: Copies the content of floating point register FRm to the system register FPUL.

Operation:

```
FLDS(float *FRm, *FPUL)          /* FLDS FRm,FPUL */
{
       *FPUL = *FRm;
        pc   += 2;
}
```

Exceptions:

None

【図32】

FMUL (Floating Point Multiply): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FMUL FRm, FRn	FRn * FRm -> FRn	1111nnnnmmmm0010	2	1	-

Description: Floating-point-arithmetically multiplies the contents of floating point registers FRm and FRn. And the result of this operation is written on the FRn.

Operation:

```
FMUL(float *FRm,*FRn)   /* FMUL FRm,FRn */
{
        clear_cause_VZ();
        if((data_type_of(FRm) == sNaN) ||
           (data_type_of(FRn) == sNaN))  invalid(FRn);
   else if((data_type_of(FRm) == qNaN) ||
           (data_type_of(FRn) == qNaN))  qnan(FRn);
  else case(data_type_of(FRm))      {
    NORM :
      case(data_type_of(FRn))       {
          PINF :
          NINF :  inf(FRn,sign_of(FRm)^sign_of(FRn));     break;
       default:   *FRn = (*FRn) * (*FRm);                 break;
      }                                  break;
    PZERO:
    NZERO:
      case(data_type_of(FRn))       {
          PINF :
          NINF :  invalid(FRn);                          break;
       default:   zero(FRn,sign_of(FRm)^sign_of(FRn));   break;
      }                                  break;
    PINF :
```

【図33】

【図33】

FNEG (Floating Point Negate): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FNEG FRn	-FRn -> FRn	1111nnnn01001101	2	1	-

【図35】

Description: Floating-point-arithmetically negates the content of floating point register FRn. And the result of this operation is written on the FRn.

Operation:

```
FNEG(float *FRn)  /* FNEG FRn */
{
  clear_cause_VZ();
  case(data_type_of(FRn))     {
      qNaN :      qnan(FRn);  break;
      sNaN :   invalid(FRn);  break;
    default:   *FRn = -(*FRm);      break;
  }
  pc += 2;
}
```

【図35】

FNEG Special Cases

FRn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
FNEG(F_n)	NEG	-0	+0	-INF	+INF	qNaN	Invaild

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

特開平9－212359

FSQRT (Floating Square Root): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FSQRT FRn	√FRn -> FRn	1111nnnn01101101	13	12	-

Description: Takes floating-point-arithmetic square root of the content of floating point register FRn. And the result of this operation is written on the FRn.

Operation:

```
FSQRT(float *FRn) /* FSQRT FRn */
{
  clear_cause_VZ();
  case(data_type_of(FRn))      {
      NORM :  if(sign_of(FRn)==0)
                        *FRn = sqrt(*FRn);
              else      invalid(FRn);      break;
      PZERO:
      NZERO:
      PINF :    *FRn = *FRn ;              break;
      NINF :   invalid(FRn);               break;
      qNaN :      qnan(FRn);               break;
      sNaN :   invalid(FRn);               break;
  }
  pc += 2;
}
```

FSQRT Special Cases

FRn	+NORM	-NORM	+0	-0	+INF	-INF	qNaN	sNaN
FSQRT (FRn)	SQRT	Invaild	+0	-0	+INF	Invaild	qNaN	Invaild

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

【図36】

【図36】

特開平9－212359

FSTS (Floating Point Store from System Register): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FSTS FPUL,FRn	FPUL -> FRn	1111nnnn00001101	2	1	-

Description: Copies the content of the system register FPUL to floating point register FRn.

Operation:

```
FSTS(float *FRn,*FPUL)  /* FSTS FPUL,FRn */
{
      *FRn  = *FPUL;
      pc  += 4;
}
```

Exceptions:
None

【図37】

FSUB (Floating Point Subtract): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FSUB FRm, FRn	FRn - FRm -> FRn	1111nnnnmmmm0001	2	1	-

Description: Floating-point-arithmetically subtracts the content of floating point register FRm from the content of floating point register FRn. And the result of this operation is written on the FRn.

Operation:

```
FSUB(float *FRm,*FRn)   /* FSUB FRm,FRn */
{
         clear_cause_VZ();
         if((data_type_of(FRm) == sNaN) ||
            (data_type_of(FRn) == sNaN))  invalid(FRn);
    else if((data_type_of(FRm) == qNaN) ||
            (data_type_of(FRn) == qNaN))  qnan(FRn);
   else case(data_type_of(FRm))      {
      NORM :
        case(data_type_of(FRn))      {
            PINF :       inf(FRn,0);        break;
            NINF :       inf(FRn,1);        break;
          default:  *FRn = *FRn - *FRm;     break;
        }                              break;
      PZERO:
        case(data_type_of(FRn))      {
            NORM :  *FRn = *FRn - *FRm;     break;
            PZERO:     zero(FRn,0);         break;
            NZERO:     zero(FRn,1);         break;
            PINF :      inf(FRn,0);         break;
            NINF :      inf(FRn,1);         break;
        }                              break;
```

【図38】

【図38】

【図３９】

【図３９】

```
            NZERO:
              case(data_type_of(ERn))      {
                   NORM :   *ERn = *ERn - *ERm;    break;
                   PZERO:
                   NZERO:       zero(ERn,0);       break;
                   PINF :        inf(ERn,0);       break;
                   NINF :        inf(ERn,1);       break;
              }                                   break;
            PINF :
              case(data_type_of(ERn))      {
                   NINF :   invalid(ERn);          break;
                 default:        inf(ERn,1);       break;
              }                                   break;
            NINF :
              case(data_type_of(ERn))      {
                   PINF :   invalid(ERn);          break;
                 default:        inf(ERn,0);       break;
              }                                   break;
        }
        pc += 2;
    }
```

【図４０】

【図４０】

FSUB Special Cases

ERm \ ERn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
NORM	SUB			+INF	-INF		
+0			-0				
-0		+0					
+INF	-INF			Invalid			
-INF	+INF				Invalid		
qNaN						qNaN	
sNaN							Invalid

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

FTRC (Floating Point Truncate and Convert to Integer): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FTRC FRm, FPUL	(long)FRm -> FPUL	1111nnnn00111101	2	1	-

【図41】

Description: Interprets the content of floating point register FRm as a floating point number value and truncates it to an integer value. And the result of this operation is written to the FPUL.

Operation:

```
#define N_INT_RENGE 0xc7000000 /* -1.000000 * 2^31 */
#define P_INT_RENGE 0x46ffffff /*  1.fffffe * 2^30 */

FTRC(float *FRm, int *FPUL)    /* FTRC FRm,FPUL */
{
  clear_cause_VZ();
  case(ftrc_type_of(FRm){
      NORM :      *FPUL = (long)(*FRm);   break;
      PINF :      ftrc_invalid(0);        break;
      NINF :      ftrc_invalid(1);        break;
  }
  pc += 2;
}
int ftrc_type_of(long *src)
{
long  abs;
      abs = *src & 0x7fffffff;
  if(sign_of(src) == 0){
        if(abs > 0x7f800000 ) return(NINF); /* NaN                */
    else if(abs > P_INT_RENGE) return(PINF); /* out of renge,+INF */
    else                       return(NORM); /* +0,+NORM          */
  }
  else {
      if(abs > N_INT_RENGE)    return(NINF); /* out of renge,+INF,NaN */
      else                     return(NORM); /* -0,-NORM             */
  }
}
```

【図41】

特開平9－212359

【図42】

```
ftrc_invalid(long *dest, int sign)
{
  set_V();
  if((FPSCR & ENABLE_V) == 0){
      if(sign == 0)      *dest = 0x7fffffff;
      else               *dest = 0x80000000;
  }
}
```

【図42】

FTRC Special Cases

FRn	NORM	+0	-0	positive out of range	negaitive out of range	+INF	-INF	qNaN	sNaN
FTRC (FRn)	TRC	0	0	Invaild +MAX	Invaild -MAX	Invaild +MAX	Invaild -MAX	Invaild -MAX	Invaild -MAX

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

FTST (Floating Point Test): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FTST/NAN FRn	(FRn==NAN)? 1:0 ->T	1111nnnn01111101	2	1	1/0

Description: Floating-point-arithmetically tests which the contents of floating point register FRn is NAN or not. And the result of this operation, true/false, is written on the T bit.

Operation:

```
FTST_NAN(float *FRn)     /* FTST/NAN FRn */
{
      clear_cause_VZ();
      case(data_type_of(FRn)) {
      NORM :
      PZERO:
      NZERO:
      PINF :
      NINF :        T = 0;      break;
      qNaN :        T = 1;      break;
      sNaN :        fcmp_invalid(1); break;
  }
      pc += 2;
}
```

FTST/NAN Special Cases

FRn	NORM	+0	-0	+INF	-INF	qNaN	sNaN
FTST/NAN (FRn)	T=0	T=0	T=0	T=0	T=0	T=1	Invaild

Denormalized value is treated as ZERO.

Exceptions:

Invalid operation

【図43】

【図43】

特開平9－212359

LDS (Load to System Register): CPU Instruction

Format	Abstract	Code	Latency	Pitch	T bit
1 LDS Rm ,FPUL	Rm->FPUL	0100nnnn01011010	1	1	-
2 LDS.L @Rm+,FPUL	@Rm->FPUL ,Rm+=4	0100nnnn01010110	2	1	-
3 LDS Rm ,FPSCR	Rm->FPSCR	0100nnnn01101010	1	1	-
4 LDS.L @Rm+,FPSCR	@Rm->FPSCR,Rm+=4	0100nnnn01100110	2	1	-

Description:

1. Copies the content of general purpose register Rm to system register FPUL.
2. Loads the content of the memory location addressed by general register Rm. And the result of this operation is written on system register FPUL. Upon successful completion, the value in Rm is incremented by 4.
3. Copies the content of general purpose register Rm to system register FPSCR. The predetermined bits of FPSCR remain unchanged.
4. Loads the content of the memory location addressed by general register Rm. And the result of this operation is written on system register FPSCR. Upon successful completion, the value in Rm is incremented by 4. The predetermined bits of FPSCR remain unchanged.

Operation:

```
#define FPSCR_MASK 0x00018c60

LDS(long  *Rm,*FPUL)            /* LDS Rm,FPUL */
{
      *FPUL = *Rm;
       pc  += 2;
}
LDS_RESTORE(long  *Rm,*FPUL)          /* LDS.L @Rm+,FPUL */
{
      if(load_long(Rm,FPUL) != Address_Error)    *Rm += 4;
       pc += 2;
}
```

【図44】

【図44】

特開平9-212359

【図４５】

【図４５】

```
LDS(long  *Rm,*FPSCR)          /* LDS Rm,FPSCR */
{
      *FPSCR = *Rm & FPSCR_MASK;
       pc    += 2;
}
LDS_RESTORE(long  *Rm,*FPSCR)        /* LDS.L @Rm+,FPSCR */
{
long  *tmp_FPSCR;
      if(load_long(Rm,tmp_FPSCR) != Address_Error){
            *FPSCR = *tmp_FPSCR & FPSCR_MASK;
            *Rm += 4;
      }
       pc += 2;
}
```

Exceptions:
　Address Error

【図４７】

【図４７】

```
{
      *Rn   = *FPSCR;
       pc += 2;
}
STS_RESTORE(long  *FPSCR,*Rn)        /* STS.L FPSCR,@-Rn */
{
long  *tmp_address = *Rn - 4;
      if(store_long(FPSCR,tmp_address) != Address_Error)
      Rn   = tmp_address;
      pc += 2;
}
```

Exceptions:
　Address Error

STS (Store from System Register): CPU Instruction

Format	Abstract	Code	Latency	Pitch	T bit
1 STS FPUL , Rn	FPUL -> Rn	0100nnnn01011010	1	1	-
2 STS.L FPUL ,@-Rn	Rn-=4,FPUL ->@Rn	0100nnnn01010110	2	1	-
3 STS FPSCR, Rn	FPSCR-> Rn	0100nnnn01101010	1	1	-
4 STS.L FPSCR,@-Rn	Rn-=4,FPSCR->@Rn	0100nnnn01100110	2	1	-

Description:

1. Copies the content of system register FPUL to general purpose register Rn.
2. Stores the content of system register FPUL into the memory location addressed by general register Rn decremented by 4. Upon successful completion, the decremented value becomes the value of Rn.
3. Copies the content of system register FPSCR to general purpose register Rn.
4. Stores the content of system register FPSCR into the memory location addressed by general register Rn decremented by 4. Upon successful completion, the decremented value becomes the value of Rn.

Operation:

```
STS(long  *FPUL,*Rn)          /* STS FPUL,Rn */
{
      *Rn  = *FPUL;
       pc += 2;
}
STS_SAVE(long  *FPUL,*Rn)           /* STS.L FPUL,@-Rn */
{
long  *tmp_address = *Rn - 4;
      if(store_long(FPUL,tmp_address) != Address_Error)
      Rn  = tmp_address;
      pc += 2;
}
STS(long  *FPSCR,*Rn)         /* STS FPSCR,Rn */
```

【図46】

【図46】

FLOAT (Floating Point Convert from Integer): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FLOAT FPUL,FRn	(float)FPUL -> FRn	1111nnnn00101101	2	1	-

Description: Interprets the content of FPUL as an integer value and converts it to a floating point number value. And the result of this operation is written on the floating point register FRn.

Operation:

```
FLOAT(int *FPUL, float *FRn)  /* FLOAT FRn */
{
      clear_cause_VZ();
      *FRn  = (float) *FPUL;
       pc += 2;
}
```

Exceptions:

None

【図48】

特開平9－212359

【図49】

FMAC (Floating Point Multiply Accumulate): Floating Point Instruction

Format	Abstract	Code	Latency	Pitch	T bit
FMAC FR0, FRm, FRn	FR0*FRm+FRn -> FRn	1111nnnnmmmm1110	2	1	-

Description: Floating-point-arithmetically multiplies the contents of floating point registers FR0 and FRm. And the result of this operation is accumulated to floating point register FRn.

Operation:

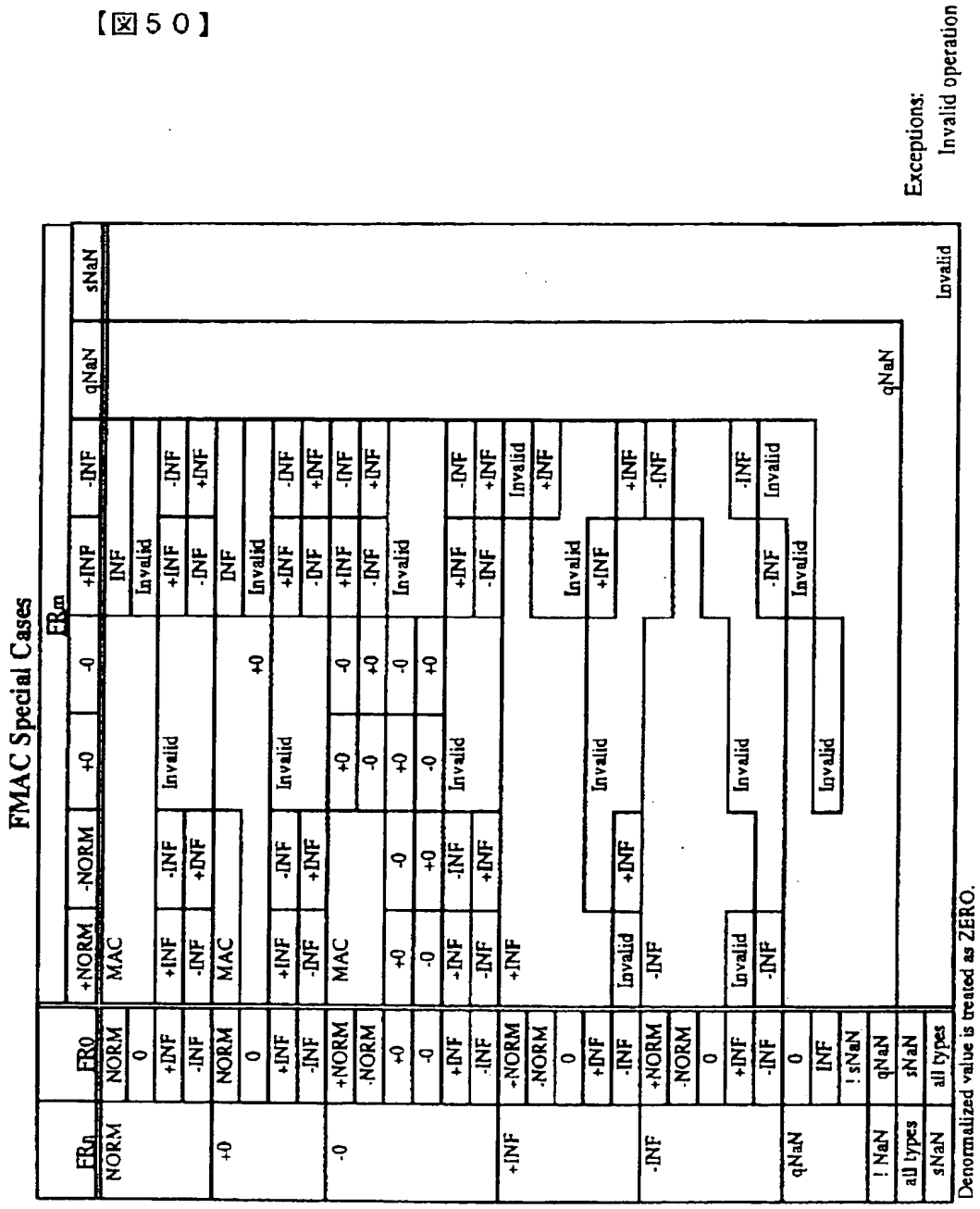
```
FMAC(float *FR0,*FRm,*_FRn)    /* FMAC FR0,FRm,_FRn */
{
long   tmp_FPSCR;
float *tmp_FMUL = *FRm;
      FMUL(FR0,tmp_FMUL);
      pc -= 2;              /* correct pc                        */
      tmp_FPSCR = FPSCR;    /* save cause field for FR0*FRm      */
      FADD(tmp_FMUL,FRn);
      FPSCR |= tmp_FPSCR;   /* reflect cause field for FR0*FRm   */
}
```

【図49】

特開平9－212359

【図５０】

【図５０】


FMAC Special Cases
FRm
FRn
FR0
+NORM
-NORM
+0
-0
+INF
-INF
qNaN
sNaN
NORM
0
+INF
-INF
MAC
Invalid
INF
+0
-0
-NORM
! NaN
all types
sNaN
Exceptions:
Invalid operation
Denormalized value is treated as ZERO.

FMOV (Floating Point Move): Floating Point Instruction

	Format	Abstract	Code	Latency	Pitch	T bit
1	FMOV FRm, FRn	FRm -> FRn	1111nnnnmmmm1100	2	1	-
2	FMOV.S @Rm, FRn	(Rm)-> FRn	1111nnnnmmmm1000	2	1	-
3	FMOV.S FRm, @Rn	FRm ->(Rn)	1111nnnnmmmm1010	2	1	-
4	FMOV.S @Rm+,FRn	(Rm)-> FRn, Rm+=4	1111nnnnmmmm1001	2	1	-
5	FMOV.S FRm,@-Rn	Rn-=4, FRm->(Rn	1111nnnnmmmm1011	2	1	-
6	FMOV.S @(R0,Rm),FRn	(R0+Rm)-> FRn	1111nnnnmmmm0110	2	1	-
7	FMOV.S FRm,@(R0,Rm)	FRm->(R0+Rn)	1111nnnnmmmm0111	2	1	-

Description:

1. Moves the content of floating point register FRm to the floating point register FRn.
2. Loads the content of the memory location addressed by general register Rm. And the result of this operation is written on the floating point register FRn.
3. Stores the content of floating point register FRm into the memory location addressed by general register Rn.
4. Loads the content of the memory location addressed by general register Rm. And the result of this operation is written on the floating point register FRn. Upon successful completion, the value in Rm is incremented by 4.
5. Stores the content of floating point register FRm into the memory location addressed by general register Rn decremented by 4. Upon successful completion, the decremented value becomes the value of Rn.
6. Loads the content of the memory location addressed by general register Rm and R0. And the result of this operation is written on the floating point register FRn.
7. Stores the content of floating point register FRm into the memory location addressed by general register Rn and R0.

Operation:

```
FMOV(float *FRm,*FRn)    /* FMOV.S FRm,FRn */
{
      *FRn  = *FRm;
       pc += 2;
}
```

【図51】

【図51】

【図５２】

【図52】

```
FMOV_LOAD(long *Rm,float *FRn)              /* FMOV @Rm,FRn */
{
      load_long(Rm,FRn);
      pc += 2;
}
FMOV_STORE(float *FRm,long *Rn)      /* FMOV.S FRm,@Rn */
{
      store_long(FRm,Rn);
      pc += 2;
}
FMOV_RESTORE(long *Rm,float *FRn)    /* FMOV.S @Rm+,FRn */
{
      if(load_long(Rm,FRn) != Address_Error)     *Rm += 4;
      pc += 2;
}
FMOV_SAVE(float *FRm,long *Rn)              /* FMOV.S FRm,@-Rn */
{
long  *tmp_address = *Rn - 4;
      if(store_long(FRm,tmp_address) != Address_Error)       Rn = tmp_address;
      pc += 2;
}
FMOV_LOAD_index(long *Rm, long *R0, float *FRn)        /* FMOV.S @(R0,Rm),FRn */
{
      load_long(&(*Rm+*R0) ,FRn) ;
      pc += 2;
}
FMOV_STORE_index(float *FRm,long *R0, long *Rn) /* FMOV.S FRm,@(R0,Rn) */
{
      store_long(FRm,&(*Rn+*R0));
      pc += 2;
}
```

Exceptions:
Address Error

フロントページの続き

(72)発明者　ケビン　イアドナト
アメリカ合衆国　カリフォルニア州
95133、サンノゼ、ガレオン　コート　678

(72)発明者　中川　典夫
東京都小平市上水本町五丁目20番１号　株式会社日立製作所半導体事業部内

(72)発明者　内山　邦男
東京都国分寺市東恋ケ窪一丁目280番地
株式会社日立製作所中央研究所内